High-Level Architecture
Object Model Template
Version 1.3

DRAFT

5 February 1998

Contents

List of Tables

1. Overview

1.1 Scope

The formal definition of the High-Level Architecture (HLA) comprises three main components: the HLA rules, the HLA interface specification, and the HLA object model template (OMT) specification (see Clause 2). This document shall provide a complete description of the essential elements of the third component of the HLA, the OMT. The other two components of the HLA formal definition are described in the following documents:

— *High-Level Architecture, Rules, Version 1.3*
— *High-Level Architecture, Interface Specification, Version 1.3*

In addition to these reference documents, other information sources relevant to developing and executing HLA federations may be found in the DMSO HLA Technical Library. The elements of the HLA technical library that are particularly relevant to HLA object model development include the following:

— *HLA Glossary*: a common set of semantics for terms used in the documents of the HLA formal definition or the HLA technical library [B6][1]
— *HLA Federation Development and Execution Process Model*: a description of the process used to build and execute HLA federations [B5]
— *HLA OMT Test Procedures***:** a set of procedures for testing compliance of an object model with the HLA OMT [B7]

Other elements of the HLA technical library may also have some relevance to HLA object model construction. Users of this document are encouraged to browse the contents of the HLA technical library to discover sources of potentially relevant information, and to gain a broader understanding of other general HLA resources.

1.2 Purpose

The purpose of the HLA is to facilitate interoperability among simulations and promote reuse of simulations and their components. To support these general goals, this document provides a specification of the HLA OMT for documenting key information about simulations and federations. More specifically, the HLA OMT provides a template for documenting HLA-relevant information about classes of simulation or federation objects and their attributes and interactions. This common template facilitates understanding and comparisons of different simulations and federations, and provides the format for a contract between members of a federation on the types of objects and interactions that will be supported across its multiple interoperating simulations. This document specifies both the type of information content required and a format for representing that content for HLA object models.

1.3 Background

1.3.1 Object model template rationale

A standardized structural framework, or template, for specifying HLA object models is an essential component of the HLA for the following reasons:

— It provides a commonly understood mechanism for specifying the exchange of data and general coordination among members of a federation
— It provides a common, standardized mechanism for describing the capabilities of potential federation members
— It facilitates the design and application of common tool sets for development of HLA object models

[1] The numbers in brackets correspond to those in the bibliography in Annex D.

HLA object models may be used to describe an individual federation member (federate), creating an HLA simulation object model (SOM); or to describe a named set of multiple interacting federates (federation), creating a federation object model (FOM). In either case, the primary objective of the HLA OMT is to facilitate interoperability among simulations and reuse of simulation components. All discussion of HLA object models in this document shall apply to both SOMs and FOMs unless explicitly stated otherwise.

1.3.2 Federation object models

During development of an HLA federation, it is critical that all federation members achieve a common understanding as to the nature or character of all required communications between participating federates. The primary purpose of an HLA FOM is to provide a specification for data exchange among federates in a common, standardized format. The content of this data includes an enumeration of all object and interaction classes pertinent to the federation, along with a specification of the attributes or parameters that characterize these classes. Taken together, the individual components of an HLA FOM establish the “information model contract” that is necessary (but not sufficient) to achieve interoperability among the federates.

1.3.3 Simulation object models

A critical step in the formation of a federation is the process of determining the composition of individual simulation systems to best meet the sponsor’s overall objectives. An HLA SOM is a specification of the intrinsic capabilities that an individual simulation could provide to HLA federations. The standard format in which SOMs are expressed facilitates determination of the suitability of simulation systems for participation in a federation.

The HLA OMT formats described in this document are generally applicable to either FOMs or SOMs. Thus, SOMs are also characterized in terms of their objects, attributes, interactions, and parameters. The primary benefit from the common utilization of the OMT formats for FOMs and SOMs is that it provides a common frame of reference for describing object models in the HLA community. In some cases, this commonality may even allow SOM components to be integrated as “piece parts” in a FOM, facilitating rapid FOM construction.

1.3.4 Relationship of HLA/OO concepts

While the HLA OMT is the standardized documentation structure for HLA object models, FOMs and SOMs do not completely correspond to common definitions of object models in object-oriented analysis and design (OOAD) methodologies. In the OOAD literature, an object model is described as an abstraction of a system developed for the purpose of fully understanding the system. To achieve this understanding, most object-oriented (OO) methodologies recommend defining several views of the “real-world” system, including complete descriptions of all static and dynamic relationships between objects and transformational (algorithmic) descriptions of the objects themselves. For HLA object models, the intended scope of the system description is much narrower, focusing specifically on requirements and capabilities for federate/federation information exchange. For SOMs, the intent is to describe the public interface of the federate in terms of an identified set of supported objects and interactions. A more complete description of how that federate is designed and functions internally (a traditional object model for object-oriented systems) should be provided by supporting design documents and other relevant documentation resources rather than by the SOM. For FOMs, the intent is to describe inter-federate information exchange within the system. A product similar to a traditional object model may be developed prior to FOM development to fully describe the real-world domain of interest (including all federation object relationships and behaviors); however, this level of system description should be used as a basis for identifying information exchange requirements rather than being part of the FOM itself.

Differences between HLA and OOAD principles and concepts also appear at the individual object level. In the OOAD literature, objects are defined as software encapsulations of data and operations (or methods). In the HLA, objects are defined entirely by the identifying characteristics (attributes) that are exchanged between federates during execution, with the behaviors and operations that affect the values of HLA object attributes kept resident in the federates. OOAD objects can be either tangible or conceptual, while HLA objects are normally intended to represent real-world entities (e.g., tank, aircraft, …). The concept of *inheritance* is fundamental to both OOAD and HLA objects, although the hierarchical structures used to define class-subclass relationships are driven by different concerns and interests. OOAD objects interact via message passing, in which one object will invoke an operation or service provided by another object, while HLA objects interact via attribute updates or interactions (a stateless, transitory exchange of information, roughly analogous to the concept of an event in the OOAD literature; see 4.3). Also, responsibility for updating HLA object attributes can be distributed among different federates in a federation (effectively distributing responsibility for maintaining the object's state across the federation), whereas OOAD objects encapsulate state locally and associate update responsibilities with operations that are closely tied to the object's class definition.

In addition to the stated semantic variations in shared terminology, other differences may also be implied as new terms are introduced throughout the OMT table descriptions. Precise definitions of all HLA terms can be found in the separate *HLA Glossary* document.

NOTE—Comments on this document should be sent by electronic mail to the Defense Modeling and Simulation Office HLA Specifications mailing address (hla_specs@msis.dmso.mil). The subject line of the message should include the OMT clause or subclause number referenced in the comment. The body of each submittal should include (1) the name and electronic mailing address of the person making the comment (separate from the mail header), (2) reference to the portion of this document that the comment addresses (by page, clause or subclause number, and paragraph number), (3) a one-sentence summary of the comment and/or issue, (4) a brief description of the comment and/or issue, and (5) any suggested resolution or action to be taken.

2. References

This standard shall be used in conjunction with the following publications. When the following standards are superseded by an approved revision, the revision shall apply:
High-Level Architecture, Rules, Version 1.3, 11 February 1998
High-Level Architecture, Interface Specification, Version 1.3, 11 February 1998

3. Definitions

ASCII	American Standard Code for Information Interchange
BNF	Backus-Naur Form
DDM	data distribution management
DM	declaration management
DoD	Department of Defense
DMSO	Defense Modeling and Simulation Office
FED	federation execution data
FOM	federation object model
HLA	High-Level Architecture
N/A	not applicable
OML	object model library

OMT	object model template
OO	object oriented
OOAD	object-oriented analysis and design
POC	point of contact
RTI	runtime infrastructure
SOM	simulation object model

4. HLA OMT components

HLA object models are composed of a group of interrelated components specifying information about classes of objects, their attributes, and their interactions. While the information content of these components can be represented in many different ways, the HLA requires that these components shall be documented in the form of tables. The template for the core of an HLA object model shall use a tabular format and shall consist of the following components:

— *Object model identification table*: to associate important identifying information with the HLA object model
— *Object class structure table*: to record the namespace of all simulation/federation object classes and to describe their class-subclass relationships
— *Interaction class structure table*: to record the namespace of all simulation/federation interaction classes and to describe their class-subclass relationships
— *Attribute table*: to specify features of object attributes in a simulation/federation
— *Parameter table*: to specify features of interaction parameters in a simulation/federation
— *Routing space table*: to specify routing spaces for object attributes and interactions in a simulation/federation
— *FOM/SOM lexicon*: to define all of the terms used in the tables

Both federations and individual simulations (federates) shall use all seven of the core OMT components when providing an HLA object model, although, in some cases, certain tables may be empty. However, all HLA object models shall contain at least one object class or one interaction class.

While federations typically will support interactions among some of the objects of their federates, some federates (such as a stealth viewer) might not be involved in interactions, so the interaction class structure table may be empty for some HLA object models. If no interactions are supported in a given object model, the parameter table will also be empty. It is expected that federates will commonly have objects with attributes of interest across the federation; in such cases, these objects and attributes shall be documented. However, a federate or an entire federation may exchange information solely via interactions, in which case its object class structure table and attribute table may be empty. The routing space table will always be empty for SOMs, and may be empty for a FOM if no data distribution management (DDM) services are being used within the federation.

The final HLA OMT component, the FOM/SOM lexicon, is essential to ensure that the semantics of the terms used in an HLA object model are understood and documented. Since there will always be at least one term in an HLA object model, there will always be at least one term defined in the lexicon, and ordinarily many more.

Any entry within any of the OMT tables may be annotated with additional descriptive information outside of the immediate table structure. This “notes” feature permits users to associate explanatory information with individual table entries as required to facilitate effective use of the data. The format for attaching a note to a table entry shall be a numerical identifier preceded by an asterisk, and enclosed by brackets. The note itself shall be distinguished by the corresponding identifier, and shall be unconstrained in terms of format. If a set of notes is defined for a given FOM or SOM, the notes shall be included as part of the object model description.

The basics of each OMT component are presented in the following separate subclauses along with a brief review of the rationale for including them in the OMT. The template format for each component is provided and described. Annex E provides a detailed specification of the allowable characters and patterns of characters for all OMT data elements. In addition, some criteria are suggested to help guide decisions on when to include specific simulation or federation features within each of these components for a specific HLA object model.

4.1 Object model identification table

4.1.1 Purpose/rationale

A key design goal for all HLA object models is to facilitate reuse. HLA object models provide information that enables inferences to be drawn regarding the reuse potential of individual federates for new applications. Reuse can also occur at the level of the object model itself. An existing FOM may provide a foundation for the development of new FOMs, or applicable components of existing SOMs may be merged to form new FOMs. In either case, to expedite reuse, it is important to include a minimum but sufficient degree of meta-level information in the object model description. For instance, when federation developers wish to pose detailed questions about how a federate or federation was constructed, including point-of-contact (POC) information within an HLA object model is extremely important. Although it may, in some instances, be appropriate to associate a wider range of meta-level information with an object model, it is always necessary to include certain key identifying information within the object model description itself.

4.1.2 Table format

The information that shall identify HLA object models shall be provided in a simple two-column table. The structure that shall be used to describe this information is provided in Table 1. The first column of this table specifies the categories of data that shall be provided in this table. The definitions of these categories shall be as follows:

— *Name*: the name assigned to the object model
— *Version*: the version identification assigned to the object model
— *Modification Date*: the date on which this version of the object model was created or last modified
— *Purpose*: the purpose for which the federate or federation was created; may also contain a brief description of key features
— *Application Domain*: the application domain to which the federate or federation applies
— *Sponsor*: the organization that sponsored the development of the federate or federation
— *POC*: the point of contact for information on the federate or federation and the associated object model; shall include an honorific (e.g., Dr., Ms., etc.) or rank, first name, and last name
— *POC Organization*: the organization with which the POC is affiliated
— *POC Telephone*: the telephone number for the POC
— *POC Email*: the email address of the POC

Table 1. Object model identification table

Object Model Indentification Table	
Category	**Information**
Name	
Version	
Date	
Purpose	
Application Domain	
Sponsor	
POC	
POC Organization	
POC Telephone	
POC Email	

The second column of this table shall be used to specify the required information. There are no particular formatting requirements associated with this information; however, the name and version number identified in the table shall match the corresponding information in the HLA object model library (OML).

4.1.3 Inclusion criteria

The categories of information specified in this table shall be included for all HLA object models. Additional types of meta-level information may be provided at the discretion of the object model developer via the HLA OML.

4.1.4 Example

Table 2 shows a simple example of the object model identification table. In this example, the *Purpose* entry is shorter (by design) than entries that would be found in this table for most practical applications. The subclauses that follow provide examples of each OMT table for the notional object model identified in this table.

4.2 Object class structure table

4.2.1 Purpose/rationale

The object class structure of an HLA object model shall be defined by a set of relations among classes of objects from the simulation or federation domain. An HLA object class is a collection of objects with certain characteristics or attributes in common. Each of the individual objects in a class shall be said to be a member (or instance) of that class.

Table 2. Object model identification table example

Object Model Indentification Table	
Category	**Information**
Name	Restaurant SOM
Version	1.0 Alpha
Date	1 Jan 1998
Purpose	To provide an example of an object model for a restaurant federate.
Application Domain	Restaurant operations
Sponsor	Federated Foods
POC	Mr. Joseph Smith
POC Organization	Joe's Place
POC Telephone	(977) 555-1234
POC Email	smithj@fedfoods.com

An HLA class structure shall be defined in terms of hierarchical relationships among classes of objects. Immediate class-to-subclass relationships shall be represented by the inclusion of the associated class names in adjacent columns of the object class structure table. Non-immediate class-to-subclass relationships shall be derived through transitivity from the immediate relations: If A is a superclass of B, and B is a superclass of C, then A is a (derived) superclass of C. Superclass and subclass play inverse roles in these relations: If A is a superclass of B, then B is a subclass of A.

Subclasses can be considered to be specializations, or refinements, of their immediate superclasses. Subclasses shall always inherit the attributes of their immediate superclass, and may possess additional characteristics to provide the desired specialization. These types of object class relationships (referred to as "is-a" relationships in the OO literature) may also be defined in terms of their instances: A class A shall be a superclass of class B only if each of the instances of class B are also instances of class A. Under this conception, it is useful to distinguish derived instances of a class from explicitly declared instances. Once an object is explicitly declared to be an instance of some object class, it shall become an implicit (or derived) instance of all the superclasses of that class. For example, if the class M1_Tank is a subclass of Tank, then an object declared to be an M1_Tank will be a derived instance of Tank. While some classes (such as Tank) might be designed for organizational purposes and not intended to have any explicitly declared instances, such "abstract" classes may still have derived instances.

A class shall be a root in a class structure if it has no superclasses in that structure. A class shall be a leaf of a class structure if it has no subclasses. If each class has at most one immediate superclass, then the class structure shall be said to have single inheritance and shall form either a tree structure or a forest of trees, depending upon whether there are one or more roots. If some classes have more than one immediate superclass, the class hierarchy shall be said to have multiple inheritance. HLA object model class hierarchies shall be represented by single inheritance (no multiple inheritance), although flat structures (with no subclasses) may also be used.
In general, federates participating in a federation execution may subscribe to object classes at any level of the class

hierarchy. By subscribing to all attributes of a specified object class, a federate is ensured of receiving all attribute value updates of attributes defined for that class and all of its superclasses for all instances of that class and all instances of its subclasses. After subscribing to an object class, a federate shall be notified by the *Discover Object* service of the existence of any instances of that class or any of its subclasses. The HLA runtime infrastructure (RTI) shall report objects as belonging to the most specific object class to which the federate is directly subscribed. If the federate subscribes to multiple levels, the RTI's discovery notification shall identify an object as an instance of the lowest-level class to which the object belongs among those subscribed by the federate.

Object classes shall provide the means for federation participants to subscribe to information about all individual instances of objects with common characteristics, such as all M1A1 tanks or F117A fighters. Classes are also essential to specifying characteristics (attributes) of simulation objects, since these are defined relative to classes of objects, not unique to individual instances. In addition, since basic RTI services (as described in the *HLA Interface Specification*) support subscriptions to object classes and their attributes by federates participating in a federation execution, the RTI requires knowledge of all object classes and their attributes if it is to perform consistency checks and to support distribution of object information by class to the federates of a federation execution.

A class hierarchy expands the capabilities of a flat classification scheme by enabling federates to subscribe to information about broad superclasses of objects, such as all tanks, all attack fighters, or even all ground vehicles, air vehicles, or sea vehicles. The existence of a class hierarchy can simplify the subscription to class information when federates are interested in broad classes of objects. The *HLA Interface Specification* supports subscription to all attributes of any class in an object class hierarchy so that federates can easily subscribe to all and only those classes of interest. An object class hierarchy also supports simplification of the specification of attributes by placing common attributes of multiple subclasses in a common superclass. Thus, class hierarchies enable simpler management of the interests of different federates in the objects and attributes involved in a federation execution.

4.2.2 Table format

The object class structure template of Table 3 shall provide a format for representing the class-subclass hierarchy of object classes. It begins with the most general object classes in the left column, followed by all their subclasses in the next column, and then a further level of subclasses. The number of intermediate columns used here shall depend upon the needs of the federation. A federation that uses a deeper hierarchy than illustrated by the template of Table 3 may add columns as needed. Finally, the most specific object classes shall be specified by enumeration in the farthest right column. For cases in which the whole class hierarchy is too deep to fit across a single page, a reference (<ref>) to a continuation table may be provided in the last column. Each object class shall have all of its subclasses specified in the next column to its right or in a continuation table referenced in that column. An example of how such a table may be filled in is provided in 4.2.4. See Annex A for a brief description of the notation that is used for specifying entries for this table.

Object class names in this table shall be defined using the ASCII character set. Although individual class names need not be unique, all object classes shall be uniquely identifiable in an HLA object model when concatenated (via dot notation) with the names of higher-level superclasses. Class names may also include other class names as parts (textual substrings) to indicate relations between classes.

Each object class in the object class structure table shall be followed by information on publication and subscription capabilities enclosed in parentheses, as designated in the template using the abbreviated variable name <*ps*>. Three basic capability levels shall be distinguished with respect to a given object class:

a) *publishable (P)*: The specified object class can be published by a federate using the *Publish Object Class* service. This also requires that a federate is capable of meaningful invocations of the *Register Object* service using this class's name.
b) *subscribable (S)*: A federate is currently capable of utilizing and (potentially) reacting to information on objects in the specified class. Qualifying for this subscription category requires the minimal capability of being able to respond appropriately to the RTI message of *Discover Object* for objects of this class.

Table 3. Object class structure table

Object Class Structure Table

<class> (<ps>)	[<class> (<ps>)]	[<class> (<ps>)]	[<class> (<ps>)] [,<class> (<ps>)]* | [<ref>]
		[<class> (<ps>)]	[<class> (<ps>)] [,<class> (<ps>)]* | [<ref>]
		...	...
		[<class> (<ps>)]	[<class> (<ps>)] [,<class> (<ps>)]* | [<ref>]
	[<class> (<ps>)]	[<class> (<ps>)]	[<class> (<ps>)] [,<class> (<ps>)]* | [<ref>]
		...	...
		[<class> (<ps>)]	[<class> (<ps>)] [,<class> (<ps>)]* | [<ref>]
	...	...	...
<class> (<ps>)	[<class> (<ps>)]	[<class> (<ps>)]	[<class> (<ps>)] [,<class> (<ps>)]* | [<ref>]
		[<class> (<ps>)]	[<class> (<ps>)] [,<class> (<ps>)]* | [<ref>]
		...	...
...	...	...	...

c) *neither publishable or subscribable (N)*: The object class is neither publishable nor subscribable by a federate.

These definitions shall apply equally to FOMs and SOMs, although an object class needs to be publishable or subscribable only by a single federate in a federation for it to be classified as *publishable* or *subscribable*, respectively, by the federation as a whole.

The *publishable* and *subscribable* capabilities are intended to identify meaningful capabilities of a federation or federate with respect to the associated object classes. Although it is difficult to formulate precise criteria for distinguishing such capabilities for all possible cases, the general intended interpretation can be characterized. An object class is publishable by a federate in this sense only if the federate is capable of somehow modeling the existence of objects in this class when it instantiates them. It is not enough to be capable of issuing calls to the cited RTI services for publication or instantiation, which any simulation might easily accomplish for any arbitrary object class. The *publishable* designation is intended to allow federates to distinguish their internal capabilities for modeling objects of the associated classes as well as their ability to share information about such objects in an HLA federation. An object class is *subscribable* by a federate only if the federate can make substantive use of instances of the class when it is notified of them by the RTI. An object class is not *subscribable* by a federate if it always ignores instantiation notices and updates for object attributes in that class. While the HLA requires that substantive capabilities underlie designations of object classes as *publishable* or *subscribable*, the detailed determination of what is meant by “substantive” for a particular FOM or SOM must be left to the discretion of their developers.

The *publishable* and *subscribable* capabilities may both be present for an object class, or various other combinations, depending on the type of class. Classes that are not *publishable* may be “abstract.” An abstract class has no explicitly declared instances since instantiations using its class name are not permitted. However, abstract classes ordinarily have “concrete” subclasses, i.e., subclasses that can be instantiated. Abstract classes can be useful for subscription purposes, simplifying some subscriptions to information about objects in their subclasses. Abstract classes can also simplify the specification of attributes by allowing common attributes of multiple object classes to be specified once in a common abstract superclass.

An individual federate shall specify its publishing and subscription capabilities in its SOM object class structure table by any of the four different combinations of publishing and subscription capabilities from the set {P, S, PS, N}. An object class may be *publishable* without being *subscribable* (*P*), may be *subscribable* without being *publishable* (*S*), or may both *publishable* and *subscribable* (*PS*) for an individual federate. In addition, some abstract object classes may be designated as neither *publishable* nor *subscribable* (*N*) if such classes facilitate the object model description.

Publication and subscription capabilities for a federation are somewhat different from those of a single federate. Whenever a federation supports publication of an object class, it must support subscription as well since it would be useless to publish an object class that could not be subscribed to within a federation. Thus, the *publishable/subscribable* capability designations for an object class in a FOM are taken from the more restricted set {*S*, *PS*, *N*}. This allows the publication and subscription capabilities recorded in a FOM to distinguish between abstract classes *(S) or (N)*, and concrete, *publishable* and *subscribable (PS)* classes.

4.2.3 Inclusion criteria

In all HLA object models, any object class that is referenced in any table within the object model shall also be included in the object class hierarchy. The criteria for designing an object class hierarchy for an HLA object model is fundamentally different for individual federates than for federations. The object class structure table of a FOM represents an agreement between the federates in a federation on how to classify object classes for the purposes of federation executions. The object class structure table of a SOM is a type of advertisement of the classes of objects that the federate can support (as *publishable* or *subscribable*) in potential federations. In neither case shall the HLA require specific object classes or object class hierarchies to appear in the object class structure table.

For individual federates, it is the intrinsic functionality (expressed as classes of objects) that the federate can offer to future HLA federations—and any object classes whose instances (and associated attribute values) may potentially represent useful information if generated by other HLA federates—that shall define the content of the object class structure table. The structure of the object class hierarchy shall be driven by how the federate supports publication and subscription of the classes in its SOM. Rich, deep SOM class hierarchies provide federates with a significant degree of flexibility in how they can support and participate in federations in the future.

For federations, it is the subscription requirements of the collective set of simulations participating in the federation that shall drive the content and structure of the object class hierarchy. While a set of concrete object classes for the most specific types of entities involved in a federation (e.g., M1 tanks and Bradley fighting vehicles) may completely satisfy the subscription requirements of some types of HLA applications, additional higher-level object classes will be needed if federates wish to be able to subscribe to object information at higher levels of abstraction (e.g., tanks, armored vehicles, or ground vehicles). For a federate to be able to subscribe to object information at a desired level of abstraction, an object class at that level of abstraction must appear in the object class structure table.
For example, suppose a federation involved both air, land, and sea forces of many specific types. If a particular federate did not require notification of the specific types of land vehicles, but did require notification of land vehicles in its area of interest, then a suitable abstract class (such as *Ground_Vehicle*) would be needed to make this possible.

While classes are clearly needed for all objects of interest in a federation, many alternative class hierarchies can be devised to cover any given set of objects. The demarcations and levels of classes selected for an HLA FOM are the result of the federation development process. Object class hierarchies that may already exist for individual SOMs may be incorporated into a FOM object class hierarchy if they meet the interests of the federation as a whole. However, since new classifications of objects may be warranted to meet federation needs that were not previously made explicit in any of their participating federates, FOM object classes and their subclass relations need not be a subset of those of the SOMs of the participating federates.

4.2.4 Example

Table 4 provides an example of how the object class structure table can be used to represent a simple system. In this case, the system being represented is a typical neighborhood restaurant. The simulation of this restaurant's operations can be considered to be a potential federate in a larger-scale federation, perhaps representing the combined, coordinated operation of a chain of restaurants. The intent of this example is not to specify a complete SOM for this system, but rather to provide partial illustrations of how the OMT tables can be used to capture relevant information about the system.

In this example, a subset of a complete object class hierarchy is shown as consisting of five object classes at the uppermost level. For this simulation, no class decomposition was necessary for the first three classes. For the fourth class, a single level of decomposition is shown resulting in five leaf classes. For the fifth class, several levels of decomposition are shown to illustrate a partial representation of the restaurant's "menu." Some of the deeper levels in this hierarchy could have been modeled as attributes (e.g., *Clam_Chowder* could have been a leaf node, with an attribute of *Type* to represent the enumerated values of *Manhattan* or *New_England*). However, the modeler in this example opted to represent the most specific food types as individual classes. In all cases in this example, abstract classes are designated as *subscribable* only, while the leaf nodes (concrete classes) are designated as both *publishable* and *subscribable*.

Table 4. Object class structure table - SOM example

Object Class Structure Table				
Customer (PS)				
Bill (PS)				
Order (PS)				
Employee (S)	Greeter (PS)			
	Waiter (PS)			
	Cashier (PS)			
	Dishwasher (PS)			
	Cook (PS)			
Food (S)	Main_Course (PS)			
	Drink (S)	Water (PS)		
		Coffee (PS)		
		Soda (PS)		
	Appetizer (S)	Soup (S)	Clam_Chowder (S)	Manhattan (PS)
				New_England (PS)
			Beef_Barley (PS)	
		Nachos (PS)		
	Entrée (S)	Beef (PS)		
		Chicken (PS)		
		Seafood (S)	Fish (PS)	
			Shrimp (PS)	
			Lobster (PS)	
		Pasta (PS)		
	Side_Dish (S)	Corn (PS)		
		Broccoli (PS)		
		Baked_Potato (PS)		
	Dessert (S)	Cake (S)		
		Ice_Cream (S)	Chocolate (PS)	
			Vanilla (PS)	

4.3 Interaction class structure table

4.3.1 Purpose/rationale

An interaction shall be defined as an explicit action taken by an object (or set of objects) in one federate that may have some effect on object(s) in a different federate. Interactions shall be specified in the interaction class structure table of HLA object models in terms of their class-subclass relationships, in much the same way that objects are described in the object class structure table. The hierarchical structure of interactions supported by this table shall be composed of relations of generalization (or specialization) between different types of interactions. For example, an engagement interaction might be specialized by air-to-ground engagements, ship-to-air engagements, and others.

This engagement interaction, then, would be said to generalize its more specific types. If there are no generalizations of interaction types for a federation or simulation, the interaction structure shall be flat, consisting of a set of unstructured interactions.

An interaction hierarchy in an HLA object model is designed to support inheritance in subscriptions. When a federate subscribes to an interaction class, using the *Subscribe Interaction Class* service, it shall receive notification of all interactions that occur during a federation execution that are identified as belonging to that class or any of its subclasses. Subscribing to an engagement interaction, for example, would result in notification of all air-to-ground engagements and ship-to-air engagements if they are subclasses of this interaction.

Interaction parameters in HLA object models shall be used to record whatever information is required to specify some features or properties of an interaction that are needed to calculate its effects by a receiving object. Examples of interaction parameters include object class names, object attributes, strings, and numerical constants. The values of all applicable interaction parameters shall be included with the interaction class name whenever federates invoke the *Send Interaction* service during a federation execution. The identification of interaction parameters, along with their class associations and details on their characteristics (such as resolution and accuracy), can be found in the parameter table of an HLA object model.

Interaction parameters may be associated with an interaction class at any level of an interaction class hierarchy. Interaction classes shall inherit the parameters defined for its superclasses. In fact, the mechanisms and rules for inheritance of interaction parameters are identical to those of object attributes. Thus, the specific placement of parameters throughout an interaction class hierarchy for a given federation shall be driven by the same types of subscription needs and requirements that drive the placement of attributes in an object class hierarchy.

Interactions are one of the principal determinants of interoperability among simulations. Interoperability ordinarily requires some consistency in the treatment of interactions afforded by the different federates in which they appear. In distributed war fighting, for example, some uniformity in treatment of engagement interactions is commonly required to ensure a fair fight between objects owned by different federates. Thus, all interactions in a FOM shall be identified, and all federates in an HLA federation shall treat the specified interactions in a uniform fashion.

In addition, the types of interactions involved in a simulation execution shall be made known to the RTI in order to support publication and subscription to their occurrences. Thus, the HLA object model shall document all of the interactions that may be sent during a federation execution so that the RTI can recognize them. Specification of the parameters of interactions in the object model serves to identify the specific information that shall be provided by any federate sending this interaction, and responded to by any federate whose objects are recipients of its effects.

4.3.2 Table format

The template that shall be used for recording object interactions for a federation or an individual federate is illustrated in Table 5. The basic format for this table shall follow the format specified earlier for the object class structure table. Thus, the most general (root) interaction classes should be specified in the leftmost column, with increasing degrees of class specificity being provided by additional columns (as necessary) in a rightward direction. Like the object class structure table, additional columns may be added to the table as needed to specify the full hierarchy, and references (<ref>) to a continuation table for deep hierarchies may be provided if desired. Interaction names in an HLA object model shall be defined using the ASCII character set, and while individual class names need not be unique, all interaction classes shall be uniquely identifiable in an HLA object model when concatenated (via dot notation) with the names of higher-level superclasses. See Annex A for a description of the syntax used for specifying entries in this table.

Similar to the <ps> designations provided in the object class structure table, the interaction class structure table also shall provide certain designations of federate/federation capabilities with respect to given classes of information. This is shown in the template as the abbreviated variable name <isr>. These designations shall be specified for

Table 5. Interaction class structure table

Interaction Class Structure Table			
<class> (<isr>)	[<class> (<isr>)]	[<class> (<isr>)]	[<class> (<isr>)] [,<class> (<isr>)]* | [<ref>]
		[<class> (<isr>)]	[<class> (<isr>)] [,<class> (<isr>)]* | [<ref>]
		...	...
		[<class> (<isr>)]	[<class> (<isr>)] [,<class> (<isr>)]* | [<ref>]
	[<class> (<isr>)]	[<class> (<isr>)]	[<class> (<isr>)] [,<class> (<isr>)]* | [<ref>]
		...	...
		[<class> (<isr>)]	[<class> (<isr>)] [,<class> (<isr>)]* | [<ref>]
	...	...	...
<class> (<isr>)	[<class> (<isr>)]	[<class> (<isr>)]	[<class> (<isr>)] [,<class> (<isr>)]* | [<ref>]
		[<class> (<isr>)]	[<class> (<isr>)] [,<class> (<isr>)]* | [<ref>]
		...	...
...	...	...	...

SOMs, while FOMs shall also include this information for uniformity. Four basic categories shall be used to indicate capabilities with respect to a given type of interaction:

— *initiates (I)*: indicates that a federate is currently capable of initiating and sending interactions of the given type
— *senses (S)*: indicates that a federate is currently capable of subscribing to the interaction and utilizing the interaction information, without necessarily being able to effect the appropriate changes to affected objects
— *reacts (R)*: indicates that a federate is currently capable of subscribing and properly reacting to interactions of the type specified by effecting the appropriate changes to any owned attributes of affected objects
— *neither initiates, senses, or reacts* (N): indicates that a federate is not currently capable of initiating, sensing, or reacting to this interaction class.

A capability of *initiates* for an interaction requires not just the ability to call the HLA *Publish Interaction Class* service for that interaction, but also the ability to model the initiation of the interaction and to invoke the HLA *Send Interaction* service for such interactions when initiated.

A federate *senses* a class of interactions if it is capable of utilizing information about such interactions via a *Receive Interaction* message after having invoked the *Subscribe Interaction Class* service. It is not enough to simply be capable of receiving such interaction messages, which any HLA-compliant federate can do; the information received in such messages must be used somehow by the federate. For example, a stealth viewer that is incapable of determining the effects of interactions might subscribe to them in order to adjust its display accordingly (e.g., to show flashes during weapons fire). Such a viewer *senses* these types of interactions, even though it never *reacts* to them, as described next.

A federate *reacts* to a class of interactions only if it has the capability for appropriately modifying the values of owned object attributes which are affected by that interaction. Naturally, not all interactions will require changes to attribute values, but instead may involve changes to internal states that affect attribute value updates. However, merely being able to reflect changes to the attribute values of objects affected by an interaction shall not represent a *reacts* capability for the interaction. Minimally, a *reacts* capability for an interaction class shall require a federate's ability to respond appropriately to the *Receive Interaction* calls from the RTI for such interactions. This response capability must include the ability to modify behaviors of affected objects as appropriate, which implies the ability to provide attribute updates for those objects.

An individual federate can support several combinations of initiating, sensing, and reacting to a given interaction class, as chosen from the set {I, S, R, IS, IR, or N}. Any interaction class specified in the SOM of a federate shall have one of these combinations of Init/Sense/React capabilities. Interaction classes specified in a FOM may choose appropriate designations from this same set, with the exception of the singular *I* designation. This is because, in a federation, at least one federate must *sense* or *react* to every interaction class that is *initiated*.

4.3.3 Inclusion Criteria

A type of interaction shall be included in a FOM whenever it can take place "across" a federation, i.e., when it is an "external" type of interaction. Common examples of such interactions in warfighting simulations include a variety of engagement interactions between platforms that may be owned by different federates. In order to document the types of interactions that federation members may need to accommodate, a FOM shall include all external interactions.

When interactions are not expected to occur across a federation, they need not appear in an HLA FOM. For example, in an engineering simulation, the interactions involved in the internal dynamics of a vehicle engine might not be part of a FOM if no other federate in the federation will interact directly with the engine component.
Since HLA SOMs are intended to be developed independently of any particular federation application, the relevance of any currently supported interaction class to future federations will generally be unknown. Thus, a simulation that supports either initiating, sensing, or reacting for an interaction class should ordinarily document

that support in its SOM if it is considered of possible interest to future federations.

4.3.4 Example

A representation of some illustrative interactions, based on the restaurant example introduced in 4.2.4, is given in Table 6. Here, the operations of the restaurant are described according to a set of interactions between customers and the employees of the restaurant. At the highest level, the restaurant federate can initiate generic customer-employee transactions. This would, for instance, allow federates that simulate management operations across the restaurant chain to subscribe to and monitor general activity levels for individual restaurants. This single high-level interaction class is then decomposed into the basic types of customer-employee interactions that occur in the restaurant. While all classes at this second level can be directly initiated by the federate, *Order_Taken*, *Food_Served*, and *Pay_Bill* are all further decomposed into more specialized classes that can also be directly initiated by the restaurant federate depending on the needs of the federation. In addition, this federate also shows the ability to sense (*S* designation) interactions of classes *Customer_Seated* and *Customer_Leaves*, possibly to monitor customer arrival activity in other restaurants within the chain, and to monitor the rate at which other restaurants can service their customers.

Table 6. Interaction class structure table - SOM example

Interaction Class Structure Table		
Customer_ Employee_ Transactions (I)	Customer_Seated (IS)	
	Order_Taken (I)	Order_Taken_ From_Kids_Menu (I)
		Order_Taken_ From_Adult_Menu (I)
	Food_Served (I)	Drink_Served (I)
		Appetizer_Served (I)
		Main_Course_Served (I)
		Dessert_Served (I)
	Customer_Pays (I)	Pay_Bill_by_ Credit_Card (I)
		Pay_Bill_by_ Cash (I)
	Customer_Leaves (IS)	

4.4 Attribute table

4.4.1 Purpose/rationale

Each class of simulation domain objects shall be characterized by a fixed set of attribute types. These attributes are named portions of their object's state whose values can change over time (such as location or velocity of a platform). Updates to the values of HLA object class attributes shall be published through the RTI and provided to other federates in a federation. An HLA object model shall document all such object attributes in the attribute table.

An HLA object model shall support representation of the following characteristics for attributes in the attribute table:

— Object class
— Attribute name
— Datatype
— Cardinality
— Units
— Resolution
— Accuracy
— Accuracy condition
— Update type
— Update rate/Condition
— Transferable/Acceptable
— Updateable/Reflectable
— Routing Space

The object class shall specify the class of objects to which the attribute applies. The attribute name shall identify the attribute. The datatype and cardinality of each attribute shall be specified. The units entries shall identify the units (such as m, km, kg) used for attribute values. A resolution characteristic shall record how finely the published values of an attribute may differ from each other. When attribute values take numeric values, a minimum possible quantitative variation in attribute value may be recorded here. When attribute values are discrete, then this fact may be recorded.

The accuracy of an attribute shall capture the maximum deviation of the attribute value from its intended value in the simulation or federation. This is often expressed as a numeric value, but may also be *perfect* for attributes that have no deviation from intended values. The accuracy condition of an attribute shall specify any conditions required for the given accuracy to hold at any given time during simulation/federation execution. It may consist of a reference to a particular type of update algorithm that determines the accuracy, or it may be an unconditional *always*.

The update type and update condition characteristics shall specify the update policies for the attribute. The transferable/acceptable characteristic shall provide an indication of whether ownership of the attribute can be transferred to or accepted from different federates. The updateable/reflectable characteristic shall be used to indicate capabilities for updating and reflecting the attribute. Finally, the routing space characteristic shall allow individual attributes to be associated with entries in the OMT routing space table.
Attributes of HLA object classes shall be specified in order to support subscription to their values by other interested members of a federation. Thus, the names of attributes and associated object classes are essential information when initializing a federation execution. Knowledge of object attributes is commonly required for effective communication between federates in a federation. In addition, while the resolutions, accuracies, and update policies of attributes represent characteristics that are not directly utilized by the RTI (as defined by the *HLA Interface Specification*), all are important to ensuring compatibility between federates in a federation. A federate operating with very low resolution, accuracy, or update rates for an attribute that it is publishing could create problems for interacting federates that are operating at higher resolutions, accuracies, or update rates. The specification of resolutions, accuracies, and update rates in an HLA FOM is a part of the FOM "contract" between federates to interoperate at the specified levels. Specification helps to ensure a common perception of the simulation space across federates in a federation, lowering the potential for inconsistency.

4.4.2 Table format

The attribute table of a FOM is designed to provide descriptive information about all object attributes represented in a federation. The template that shall be used for the attribute table is provided by Table 7. See Annex A for a brief description of the syntax used for specifying entries in this table. The first column (Object) shall list the names of object classes. Object class names shall include, using dot notation, the parentage (superclasses) of the class to the depth necessary to identify uniquely the class in this table. The classes themselves may be chosen from any level of generality in the class structure hierarchy. In general, to reduce redundancy, attributes should be specified for classes at the highest point in the hierarchy to which they generally apply. For example, if all air vehicles have an attribute of minimum turn radius at maximum speed, some redundancy can be avoided if this attribute is specified just once for the entire class of *Air_Vehicle*. Given that all object classes inherit the attribute types of their superclasses, the subclasses of *Air_Vehicle*, such as *Fixed_Wing* and *Rotary_Wing,* also have this attribute with its specified characteristics. When a subclass requires a revision to any inherited attribute characteristic, a new attribute shall be defined for the subclass with the required characteristics.

The second column (Attribute) shall list the attributes of the specified object class. The names assigned to attributes of any particular object class shall be defined using the ASCII character set, and shall not duplicate (overload) the names of attributes of any higher-level superclasses. There may be many attributes for a single object class.

The Datatype column shall be used to reference the datatype of the attribute. This datatype may be chosen from the list of permissible base attribute/parameter types (as described in Annex B), or it may be a user-defined datatype. User-defined datatype names shall be different from (shall not overload) the names of the base attribute types. When an attribute can take on a value of one of the base datatypes, that datatype should be recorded in the Datatype column. For attributes that can take on a finite discrete set of possible values, the enumerated datatype table (4.6.2) shall be used to characterize the datatype. Once the enumerated type has been defined, it can be recorded in the Datatype column of such attributes. For attributes whose values cannot be described by a single base or enumerated datatype, the complex datatype table (4.6.3) shall be used to define a new type by aggregating other datatypes into a structure. Once defined in the complex datatype table, the complex type can be recorded in the Datatype column of such attributes.

The Cardinality column shall be used to record the size of an array or sequence. A designation of *1+* in this column shall allow for unbounded sequences. Cardinalities of multi-dimensional arrays shall include the sizes of every dimension listed in their normal order of precedence. A one (1) shall be entered in this column for any attribute that is composed of a single instance of the datatype indicated in the Datatype column. Values other than 1 in this column shall indicate that the attribute is an array or sequence of the datatype indicated in the Datatype column.

The Units, Resolution, Accuracy, and Accuracy Condition columns are not applicable if the datatype for an attribute is either enumerated or both complex and heterogeneous. These columns are not applicable because these classes of information are either unnecessary (for enumerated datatypes), or are recorded for the individual fields of complex datatypes in the complex datatype table. For these and other datatypes in which units, resolution, and accuracy information do not apply (e.g., strings), the designator N/A for “not applicable” shall be entered.

The Units column shall contain the units (e.g., m, km, kg) used for each attribute whenever such units exist. Any units entered in this column shall specify the units of the entries in the Resolution and Accuracy columns that follow it.

The Resolution column may have different kinds of entries, depending upon the kind of attribute. For attributes of scalar numerical measures, the resolution column may contain a single-dimensioned numeric entry for each row of the table. This value may specify the smallest resolvable value separating attribute values that can be discriminated. However, when such attributes or parameters are stored in floating point datatypes, their resolution so defined might vary with the magnitude of the attribute value. Hence, in these cases and others, a better sense of the resolution may be conveyed by the datatype.

The Accuracy column is intended to capture the maximum deviation of the attribute value from its intended value in the federate or federation. This is ordinarily expressed as a dimensioned value, but it may also be *perfect* for many discrete or enumerated attributes. The Accuracy Condition column shall contain any conditions required for the given accuracy to hold in a given simulation or federation execution. It may consist of reference to a particular type of update algorithm that determines the accuracy, or it may be an unconditional *always*.

Table 7. Attribute table

Attribute Table

Object	Attribute	Data-type	Cardi-nality	Units	Resolution	Accuracy	Accuracy Condition	Update Type	Update Condition	T/A	U/R	Routing Space
<class>	<attribute>	<datatype>	[<size>]	<units>	<resolution>	<accuracy>	<condition>	<type>	<rate> | <condition>	<ta>	<ur>	<r_space>
	<attribute>	<datatype>	[<size>]	<units>	<resolution>	<accuracy>	<condition>	<type>	<rate> | <condition>	<ta>	<ur>	<r_space>
	...	...	...	...	...	...	...	...	...	...	...	...
<class>	<attribute>	<datatype>	[<size>]	<units>	<resolution>	<accuracy>	<condition>	<type>	<rate> | <condition>	<ta>	<ur>	<r_space>
	<attribute>	<datatype>	[<size>]	<units>	<resolution>	<accuracy>	<condition>	<type>	<rate> | <condition>	<ta>	<ur>	<r_space>
	...	...	...	...	...	...	...	...	...	...	...	...
<class>	<attribute>	<datatype>	[<size>]	<units>	<resolution>	<accuracy>	<condition>	<type>	<rate> | <condition>	<ta>	<ur>	<r_space>
...	...	...	...	...	...	...	...	...	...	...	...	...

T/A - Transferable/Acceptable

U/R - Updateable/Reflectable

The Update Type and Update Condition columns shall record the update policies for an attribute. The update type may be specified as *static*, *periodic*, or *conditional*. When the update type is *periodic*, a rate of number of updates per time-unit may be specified in the Update Condition column. Attributes with a *conditional* update type may have the conditions for update specified in the Update Condition column.

The Transferable/Acceptable (T/A) column is handled somewhat differently for simulations and federations. In a federation, if an attribute is transferable from a federate, it must be acceptable by some federate in the federation. But a single federate may be able to transfer ownership of an attribute without being able to accept handoff of attribute ownership from another federate. The basic alternatives for the Transferable/Acceptable column shall be as follows:

— *Transferable (T)*: A federate is currently capable of publishing and updating attributes of the type specified for the object class, and can transfer ownership of the attribute to another simulation using the HLA ownership management services.
— *Acceptable (A)*: A federate is currently capable of accepting ownership of this attribute from another federate, including the capability for meaningful continuation of attribute updates.
— *Not transferable or acceptable (N)*: A federate is not currently capable of either transferring ownership of this attribute to another federate or accepting ownership of this attribute from another federate.

For an attribute of a SOM, the transferable/acceptable variable <*ta*> shall take any of the values from the set {T, A, TA, N}. In a FOM, the only valid entries in this column for federation attributes are *TA* or *N*.

The Updateable/Reflectable (U/R) column of the attribute table shall be used to identify the current capabilities of a federate with respect to attribute updating and reflection. Two basic categories shall be used to indicate capabilities with respect to a given attribute:

— *Updateable (U)*: The federate is currently capable of publishing and updating attributes of the type specified for the object class using the *Publish Object Class* and *Update Attribute Values* services.
— *Reflectable (R)*: The federate is currently capable of accepting changes to this type of attribute for objects in the specified object class for values provided from calls to the *Reflect Attribute Values* service from the RTI.

For an attribute of a SOM, the updateable/reflectable variable <*ur*> in the attribute table may take any of three different combinations of capabilities for updating and reflecting, as designated by their abbreviations {U, R, UR}. In a SOM, any listed attribute shall be either updateable or reflectable or both. For federations, the appropriate entry shall be *UR* since all attributes in a FOM should be both updateable and reflectable.

The Routing Space column shall record the association of an object attribute with a routing space if a federation is using DDM services. The column shall contain a routing space name from the routing space table. Object attributes can be individually associated with routing spaces, although no more than one routing space shall be specified for each attribute. When attributes are sent via *Update Attribute Values*, they are subject to filtering through regions in the routing space. For SOMs, or for FOMs in which the federation is not using DDM services, N/A shall be entered into this column.

4.4.3 Inclusion Criteria

All object attributes whose values can be exchanged during the course of an HLA federation execution shall be documented in the attribute table of a FOM. All attributes that can be either updated or reflected by an individual federate shall be documented in the attribute table of its SOM.

In some object model descriptions, it may be desirable to document the capability or intent to transfer the privilege of deleting the instantiation of an object class from one federate to another. In this case, the attribute *privilegeToDeleteObject*, which is automatically created by the RTI when instantiating an object, should be included in the attribute table to document the applicable transferability characteristics. If omitted from the table, this privilege shall be assumed to be neither transferable nor acceptable.

4.4.4 Example

Table 8 shows examples of attributes from the restaurant application described in 4.2.4. In the first entry, the *Employee* object is characterized according to the four attributes shown in the table. The datatypes specified for each of the first three attributes were selected from the list of attribute/parameter basetypes (Annex B), while the datatype of the fourth attribute is user defined. As with all user-defined datatypes, the indicator *N/A* is placed in the Units, Resolution, Accuracy, and Accuracy Condition columns. Each of these four attributes is updated conditionally except for the *Years_of_Service* attribute, which is updated periodically (yearly) on the employee's start date anniversary. The Update Condition column for the *Pay_Rate* attribute is annotated with an explanatory "note" as described earlier in Clause 4. As with all of the attributes shown in this example, the attributes of *Employee* are assumed transferable, acceptable, updateable, and reflectable.

The *Waiter* subclass of *Employee* is shown with three attributes. These are in addition to the four inherited attributes from its superclass. Each of the first two attributes, *Efficiency* and *Cheerfulness*, is intended to represent a numeric score (performance measure), that is assigned to the employee at yearly performance reviews. The third attribute is intended to represent the state of the employee (the task he/she is performing) at any point in time during restaurant operations. This attribute is characterized by an enumerated datatype that is described in a separate table.

In the final two entries, the *Drink* and *Soda* classes are each shown with a single attribute. Since this example is a SOM, *N/A* has been entered for each attribute in the Routing Space column. Had this example been a FOM, both the *Number_Cups* (inherited) and *Flavor* attributes of the *Soda* class may have been associated with the *Bar_Order* routing space shown in the routing space table example in 4.7.4.

4.5 Parameter table

4.5.1 Purpose/rationale

Most interaction classes will be characterized according to a list of one or more interaction parameters. Interaction parameters shall be used to associate relevant and useful information with classes of interactions. While some subscribers of interactions may not require all associated parameters, others will need the full set of parameter-specified information to support calculation of new attribute values for objects affected by the interaction. For every interaction class identified in the interaction class structure table, the full set of parameters associated with that interaction class shall be described in the parameter table.

An HLA object model shall support representation of the following characteristics for each parameter in the parameter table:

— Interaction class
— Parameter name
— Datatype
— Cardinality
— Units
— Resolution
— Accuracy
— Accuracy condition

Table 8. Attribute table - SOM Example

Attribute Table												
Object	**Attribute**	**Datatype**	**Cardinality**	**Units**	**Resolution**	**Accuracy**	**Accuracy Condition**	**Update Type**	**Update Condition**	**T/A**	**U/R**	**Routing Space**
Employee	Pay_Rate	Float	1	Cents/Hour	1	perfect	always	conditional	Merit Increases [*1]	TA	UR	N/A
	Years_of_Service	Short	1	Years	1	perfect	always	periodic	1/year, on Anniversary	TA	UR	N/A
	Home_Number	String	1	N/A	N/A	perfect	always	conditional	Employee Request	TA	UR	N/A
	Home_Address	Address_Type	1	N/A	N/A	N/A	N/A	conditional	Employee Request	TA	UR	N/A
Waiter	Efficiency	Short	1	N/A	1	perfect	always	periodic	Performance Review	TA	UR	N/A
	Cheerfulness	Short	1	N/A	1	perfect	always	periodic	Performance Review	TA	UR	N/A
	State	Waiter_Tasks	1	N/A	N/A	N/A	N/A	conditional	Work Flow	TA	UR	N/A
Drink	Number_Cups	Short	1	Cups	1	perfect	always	conditional	Customer Request	TA	UR	N/A
Soda	Flavor	Flavor_Type	1	N/A	N/A	N/A	N/A	conditional	Customer Request	TA	UR	N/A

[*1] Merit raises are not provided according to any regular time interval, but are provided on a supervisor's recommendation based on evidence of exceptional effort and performance.

The interaction class shall specify the class of interactions to which the parameter applies. The parameter name shall identify the parameter. The datatype and cardinality of each parameter shall be specified. The units (such as m, km, kg) used for parameter values shall be specified. A resolution characteristic shall record how finely the values of a given parameter may differ from each other. When parameter values take numeric values, a minimum possible quantitative variation in parameter value may be recorded here. When parameter values are discrete, then this fact may be recorded. The accuracy of a parameter shall capture the maximum deviation of the parameter value from its intended value in the simulation or federation. This is often expressed as a numeric value, but may also be *perfect* for parameters that have no deviation from intended values. The accuracy condition of a parameter shall specify any conditions required for the given accuracy to hold at any given time during simulation/federation execution. This entry may be an unconditional *always* if appropriate.

Unlike object attributes, interaction parameters may not be subscribed to on an individual basis. This implies that (for federations) routing space information shall be specified at the interaction class level rather than at the individual parameter level. In addition, the use of inheritance allows individual parameters to be specified at the level of the interaction class hierarchy that meets the subscription needs of the federation. The names and placement of parameters throughout the interaction class hierarchy shall thus represent critical initialization data for a federation execution. The parameter characterization data (e.g., units, resolutions, accuracies) shall be specified in order to avoid inconsistent treatment of this data between federates.

4.5.2 Table format

The parameter table of an HLA object model is designed to provide descriptive information about all interaction parameters represented in a federation. The template that shall be used for the parameter table is provided by Table 9. See Annex A for a brief description of the syntax used for specifying entries in this table.

The first column (Interaction) shall list an interaction class name. Interaction class names shall include, using dot notation, the parentage (superclasses) of the class to the depth necessary to uniquely identify the class in this table. The classes may be chosen from any level of generality in the class structure hierarchy. In general, to reduce redundancy, parameters should be specified for classes at the highest point in the hierarchy in which they represent useful information, although this is not required. For example, if all weapon firings include a parameter that defines the infrared signature of the platform at the time the firing occurs, some redundancy will be avoided if this parameter is specified just once at the uppermost level of a *Weapon_Fires* class. Given that all interaction subclasses inherit the parameter types of their superclasses, the subclasses of *Weapon_Fires*, such as *Tank_Fires* and *TEL_Fires,* also have this parameter with its specified characteristics. When a subclass requires a revision to any inherited parameter characteristic, a new parameter shall be defined for the subclass with the required characteristics.

The second column (Parameter) shall list the parameters of an interaction. The names assigned to parameters of any particular interaction class shall be defined using the ASCII character set, and shall not duplicate (overload) the names of parameters of any higher-level superclasses. There may be many parameters for a single interaction class.

The Datatype column shall be used to reference the datatype of the parameter. This datatype may be chosen from the list of permissible base attribute/parameter types (as described in Annex B), or it may be a user-defined datatype. User-defined datatype names shall be different from (shall not overload) the names of the base parameter types. When a parameter can take on a value of one of the base datatypes, that datatype shall be recorded in the Datatype column. For parameters that can take on a finite discrete set of possible values, the enumerated datatype table (4.6.2) shall be used to characterize the datatype. Once the enumerated type has been defined, it can be recorded in the Datatype column of such parameters. For parameters whose values cannot be described by a single base or enumerated datatype, the complex datatype table (4.6.3) shall be used to define a new type by aggregating other datatypes into a structure. Once defined in the complex datatype table, the complex type can be recorded in the Datatype column of such parameters.

Table 9. Parameter table

Parameter Table								
Interaction	**Parameter**	**Data-type**	**Cardi-nality**	**Units**	**Resolution**	**Accuracy**	**Accuracy Condition**	**Routing Space**
<interaction>	<parameter>	<datatype>	[<size>]	<units>	<resolution>	<accuracy>	<condition>	<r_space>
	<parameter>	<datatype>	[<size>]	<units>	<resolution>	<accuracy>	<condition>	
	...	...	...	...	...	...	...	
<interaction>	<parameter>	<datatype>	[<size>]	<units>	<resolution>	<accuracy>	<condition>	<r_space>
	<parameter>	<datatype>	[<size>]	<units>	<resolution>	<accuracy>	<condition>	
	...	...	...	...	...	...	...	
<interaction>	<parameter>	<datatype>	[<size>]	<units>	<resolution>	<accuracy>	<condition>	<r_space>
...	...	...	...	...	...	...	...	...

The Cardinality column shall be used to record the size of an array or sequence. A designation of *1+* in this column shall allow for unbounded sequences. Cardinalities of multi-dimensional arrays shall include the sizes of every dimension listed in their normal order of precedence. A one (1) shall be entered in this column for any parameter that is composed of a single instance of the datatype indicated in the Datatype column. Values other than one in this column shall indicate that the parameter is an array or sequence of the datatype indicated in the Datatype column.

The Units, Resolution, Accuracy, and Accuracy Condition columns are not applicable if the datatype for a parameter is either enumerated or both complex and heterogeneous. The reason these columns are not applicable is that these classes of information are either unnecessary (for enumerated datatypes), or are recorded for the individual fields of complex datatypes in the complex datatype table. For these and other datatypes in which units, resolution, and accuracy information do not apply (e.g., strings), the designator *N/A* for "not applicable" shall be entered.

The Units column shall contain the units (e.g., m, km, kg) used for each parameter whenever such units exist. Any units entered shall specify the units of the entries in the Resolution and Accuracy columns that follow this column.

The Resolution column may have different kinds of entries, depending upon the kind of parameter. For parameters of scalar numerical measures, the resolution column may contain a single dimensioned numeric entry for each row of the table. This value may specify the smallest resolvable value separating parameter values that can be discriminated. However, when such parameters are stored in floating point datatypes, their resolution so defined might vary with the magnitude of the parameter value. Hence, in these cases and others, a better sense of the resolution may be conveyed by the datatype.

The Accuracy column is intended to capture the maximum deviation of the parameter value from its intended value in the federate or federation. This is ordinarily expressed as a dimensioned value, but may also be *perfect* for many discrete or enumerated parameters. The Accuracy Condition column shall contain any conditions required for the given accuracy to hold in a given simulation or federation execution. It may consist of reference to a particular type of update algorithm that determines the accuracy, or it may be an unconditional *always*.

The Routing Space column shall record the association between an interaction class and a routing space if the federation is using DDM services. The column shall contain a routing space name from the routing space table. This shall indicate that whole interactions of this class will be subject to filtering through regions in the routing space when the interactions are sent via *Send Interaction with Region*. No more than one routing space shall be specified for each interaction class. For SOMs, or for FOMs in which the federation is not using DDM services, *N/A* shall be entered into this column.

4.5.3 Inclusion criteria

In federations, any information elements that can be provided and associated with a given interaction class (by the class publishers) that are deemed to be useful to subscribers of that interaction class shall be included and documented as interaction parameters in the parameter table. For individual federates, SOM developers shall associate with their publishable interaction classes whatever information they feel will be needed by subscribers of their interactions to calculate changes to the values of affected attributes. In addition, SOM developers shall determine what types of information shall be included with interaction classes that the federate may subscribe to in order for that federate to calculate associated effects.

4.5.4 Example

Table 10 shows an example of parameters from the restaurant application described in 4.2.4. Here, the *Main_Course_Served* interaction has three parameters associated with it. In this case, two of the three parameters are user-defined datatypes. Since the Units through the Accuracy Condition columns do not apply for user-defined datatypes, only the Datatype and Cardinality columns have entries for these first two attributes. The third parameter uses a boolean datatype (yes or no) to reflect whether the meal was served in a reasonable amount of time.
Since this example is meant to illustrate a SOM, *N/A* has been entered in the Routing Space column. However, if this example was a FOM, this interaction could have been associated with the *Server_Order* routing space from the routing space table example in 4.7.4 as a means of filtering on waiter identification numbers.

4.6 Attribute table/parameter table subcomponents

4.6.1 Purpose/rationale

While both the attribute table and the parameter table provide columns for datatype specifications, they do not provide definitive guidance for specifying complex datatypes. This subclause describes additional table formats for complex datatypes as well as for enumerated datatypes to better document their structure and content. These tables shall be used for situations in which a federation or federate implements the attribute or parameter datatypes for which the tables are designed.

Table 10. Parameter table - SOM example

Parameter Table								
Interaction	**Parameter**	**Data-type**	**Cardi-nality**	**Units**	**Resolution**	**Accuracy**	**Accuracy Condition**	**Routing Space**
Main_Course_ Served	Temperature_ OK	Temp_Type	1	N/A	N/A	N/A	N/A	N/A
	Accuracy_ OK	Accur_Type	1	N/A	N/A	N/A	N/A	
	Timeliness_ OK	Boolean	1	N/A	1	perfect	always	

4.6.2 Enumerated datatype table

Table 11 illustrates the format which shall be used for the enumerated datatype table. The first column shall define the identifier (or name) for the enumerated datatype, while the second column shall provide the specific enumerated values that the identifier can assume. For example, one potential identifier for an enumerated datatype might be *affiliation*, with the values of *red*, *blue*, and *neutral* representing valid enumerators. The Representation column of the enumerated datatypes table shall allow the federation to define the agreed-upon numerical value for the specific enumerators. Each identifier name shall appear as an entry in the Datatype column of the OMT attribute table or parameter table, as was discussed in 4.4.2 and 4.5.2. See Annex A for a brief description of the general format used for specifying the types of entries in this table.

An example of the use of the enumerated datatype table is provided in Table 12. Here, the user-defined *Waiter_Tasks* and *Flavor_Type* datatypes specified in the earlier attribute table example (4.4.4) are each characterized according to several different enumerations. In the first case, each enumeration represents a state that a waiter can be in at any point in time during restaurant operations. In the second case, each enumeration represents a different flavor of soda. The numerical representation of the enumerations need not be given in any particular order, but shall be documented to avoid inconsistent representations among different federates in a federation.

4.6.3 Complex datatype table

Table 13 illustrates the format which shall be used for the complex datatype table. The first column, Complex Datatype, shall provide the identifier, or name, of the user-defined complex datatype. Complex data type identifiers shall match a datatype entry from the attribute table, the parameter table, or from the complex datatype table itself. The next column, Field Name, shall provide the means to identify each individual field within the complex datatype. For example, a complex datatype representing location (with *Location* as its identifier) might have three sub-rows with the field names of *X*, *Y*, and *Z* (for rectangular coordinates). Alternately, two sub-rows with the field names of *Lat* and *Long* might be used. The actual specification of the fields associated with a particular identifier shall be entirely driven by the requirements of the federate or federation.

Table 11. Enumerated datatype table

Enumerated Datatype Table		
Identifier	**Enumerator**	**Representation**
<datatype>	<enumerator>	<integer >
	. . .	. . .
<datatype>	<enumerator>	<integer>
	. . .	. . .
. . .	. . .	. . .

Table 12. Enumerated datatype table - SOM example

Enumerated Datatype Table		
Identifier	**Enumerator**	**Representation**
Waiter_Tasks	Taking_Order	1
	Serving	2
	Cleaning	3
	Calculating_Bill	4
	Other	5
Flavor_Type	Cola	101
	Orange	102
	Root_Beer	103
	Cream	104

Table 13. Complex datatype table

Complex Datatype Table

Complex Datatype	Field Name	Datatype	Cardinality	Units	Resolution	Accuracy	Accuracy Condition
<complex datatype>	<field>	<datatype>	<size>	<units>	<resolution>	<accuracy>	<condition>
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	<field>	<datatype>	<size>	<units>	<resolution>	<accuracy>	<condition>
...	...	...	...	...	...	...	...
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
<complex datatype>	<field>	<datatype>	<size>	<units>	<resolution>	<accuracy>	<condition>
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.

The remaining six fields in the complex datatype table shall be identical to the corresponding columns in the attribute table and parameter table (4.4.2, 4.5.2). The intent is to capture these classes of information for each field within the complex data structure. For complex attributes, this allows certain characteristics common to all fields (update type/condition, transferable/acceptable, updateable/reflectable) to be specified at the composite level, while characteristics distinctive of the individual fields of an attribute (units, resolution, etc.) are specified at this lower level.

The complex datatype table may also include the names of other complex datatype identifiers within the Datatype column for individual field names. This allows users to build "structures of data structures" according to the needs of their federate or federation. See Annex A for a brief description of the general format used in specifying the types of entries permitted in this table.

An example of the use of the complex datatype table is provided in Table 14. The first complex datatype (*Address_Type*) is shown as consisting of four fields, each identified as an *String* datatype. Each of the other two complex datatypes (*Temp_Type* and *Accur_Type*) consists of three boolean fields. The intent is to specify for each *Main_Course* (composed of one *Entree* and two instances of *Side_Dish*) whether the waiter served exactly what the customer ordered (*Accuracy_OK* parameter) and whether the food was the right temperature (*Temperature_OK* parameter). This information is used by all recipients of the *Main_Course_Served* interaction to determine the value of the customer attribute *Satisfaction*.

4.7 Routing space table

4.7.1 Purpose/rationale

All federations shall use declaration management (DM) services to enable the flow of object attribute and interaction data between federates. In addition to enabling the flow of this data, the DM services also limit the delivery of some data on the basis of object class, interaction class, and attribute name. This reduction of data may be insufficient to meet the needs of federations with large numbers of federates, large numbers of objects and/or interactions in classes, and/or large numbers of attribute updates per object. Such federations will need to use DDM services to further reduce the volume of data delivered to federates. When DDM services are required for a federation, a common framework for specifying the data distribution model for the federation is essential for the following reasons:

— It provides a commonly understood mechanism for specifying the exchange of public data and DDM coordination among members of a federation.
— It facilitates the design and application of common tool sets for specification of federation DDM needs.

Routing spaces are the most fundamental DDM concept. A routing space is a multidimensional coordinate system through which federates either express an interest in receiving data or declare their intention to send data. These intentions are expressed through

— *Subscription regions*: bounding routing space coordinates that narrow the scope of interest of the subscribing federate
— *Update Regions*: bounding routing space coordinates that are guaranteed to enclose an object's location in the routing space

During development of an HLA federation, it is critical that all federation members achieve a common understanding of DDM routing spaces and their semantics, and agree to a common set of routing space specifications. These agreements are necessary for federates to filter object attribute updates and interactions in a correct and consistent manner. This includes the specification of routing space names, the data to be routed through them, the names of the routing space dimensions, and all datatype and normalization information associated with each dimension. The dimensions form the parameter space in which update and subscription region boundaries are specified by the federates to the RTI. By having this agreement about the meaning of routing space dimensions enforced at the federate level, the RTI can calculate the intersections of update and subscription regions efficiently

Table 14. Complex datatype table - SOM example

Complex Datatype Table							
Complex Datatype	**Field Name**	**Datatype**	**Cardinality**	**Units**	**Resolution**	**Accuracy**	**Accuracy Condition**
Address_Type	Street	String	1	N/A	N/A	perfect	always
	City	String	1	N/A	N/A	perfect	always
	State	String	1	N/A	N/A	perfect	always
	Zip	String	1	N/A	N/A	perfect	always
Temp_Type	Entree	Boolean	1	N/A	1	perfect	always
	Vegie_1	Boolean	1	N/A	1	perfect	always
	Vegie_2	Boolean	1	N/A	1	perfect	always
Accur_Type	Entree	Boolean	1	N/A	1	perfect	always
	Vegie_1	Boolean	1	N/A	1	perfect	always
	Vegie_2	Boolean	1	N/A	1	perfect	always

without having to understand the semantics of the dimensions. The routing space table records all the elements necessary to fully specify this agreement in a standardized format. Some information described in this table is explicitly required for generation of federation execution data (FED) files, while other table elements are simply intended to ensure a common understanding of data-routing requirements among federation developers.

4.7.2 Table format

The template that shall be used for recording routing spaces is illustrated in Table 15. The highest-level structure that shall be recorded in the table is the routing space, which is provided in the first column. Its name may be used to indicate the type of filtering done by the routing space, e.g. a routing space could be named *Position* if it were to be used to filter battlefield entities based on geographic location.

The next column shall specify the name of the routing space dimension. Each routing space may have one or more dimensions, each representing a specific characteristic of the routing space. For example, the *Position* routing space might have a single dimension named *Altitude_Limits*, indicating the desire to filter location information based on altitude.

The next column shall specify the datatype for the dimension. The type element for each dimension shall be one of the basetypes specified in Annex B, or a user-defined datatype, specified in the enumerated datatype table.

The next column shall provide the range/set element, which shall specify all of the possible values within the dimension. For dimensions whose types are *float, double, short, unsigned short, long, unsigned long,* or *char*, the range element shall be specified as a pair of maximum and minimum values separated by a dash, and enclosed in square brackets or parentheses to indicate closed or open intervals, respectively. For all other dimension types, the set element shall be specified as a list of all possible values separated by commas.

The units element is informational and is intended to describe the units associated with the dimension range/set. This element may be marked *N/A* when such information is implied or is considered "unitless".

The final element specified for each dimension is its normalization function, which shall be specified in the final column. Normalization functions shall be used to map a subscription/update region's bounding coordinates to the fixed routing space dimension associated with an RTI. Since the mapping of a point into a dimension may be non-linear, and since the number of possible normalization functions is quite large, the federates shall perform this mapping themselves. To ensure that the federation will exhibit correct semantics, federation participants shall agree about the normalization functions that they will use for each dimension of each routing space. Example normalization functions are listed in Annex C.

4.7.3 Inclusion Criteria

Federations that require more data reduction than the class-based filtering provided by DM services shall use the routing space table to document federation-wide agreements on the semantics and use of DDM routing spaces. The routing space name and dimension names from this table shall be translated into entries in the FED file for use by the RTI. The type, range/set, units, and normalization function information is guidance to federates on the consistent use of routing spaces to maintain correct DDM semantics across the federation.

Table 15. Routing space table

Routing Space Table					
Routing Space	**Dimension**	**Dimension Type**	**Dimension Range/Set**	**Range/Set Units**	**Normalization Function**
<r_space>	<dimension>	<type>	<range/set>	<units>	<n_function>
	<dimension>	<type>	<range/set>	<units>	<n_function>
	<dimension>	<type>	<range/set>	<units>	<n_function>
...	...	...	...	...	...
<r_space>	<dimension>	<type>	<range/set>	<units>	<n_function>
	<dimension>	<type>	<range/set>	<units>	<n_function>
...	...	...	...	...	...

4.7.4 Example

Table 16 illustrates an example of the use of the routing space table. The routing space name given first in this example is *Bar_Order*, and will be used as a means of filtering drink orders. This routing space has an enumerated dimension *(Soda_Flavor*), and an integer dimension *(Bar_Quantity)*. The second routing space shown in this example filters on the IDs of the waiters that are in the range 1-20. No units are specified for dimension *Soda_Flavor* since it is an enumerated set. Units are not specified for *Bar_Quantity* or *Waiter_ID* because their units are implied by the dimension name and range/set, or are unitless. The *Soda_Flavor*'s *linear_enumerated* normalization function (see Annex C) would map *Cola* to the lowest RTI dimension value, *Orange* to the middle value, and *Root_Beer* to the highest value. Both *Bar_Quantity* and *Waiter_ID* values would be mapped at uniform intervals across the RTI dimension range.

Table 16. Routing space table example

Routing Space Table					
Routing Space	**Dimension**	**Dimension Type**	**Dimension Range/Set**	**Range/Set Units**	**Normalization Function**
Bar_Order	Soda_Flavor	Flavor_Type	Cola,Orange, Root Beer	N/A	linear_enumerated (Flavor)
	Bar_Quantity	short	[1-25]	N/A	linear (Number_Cups)
Server_Order	Waiter_ID	short	[1-20]	N/A	linear (Waiter_ID)

5. FOM/SOM lexicon

5.1 Purpose/rationale

If interoperability among simulations is to be achieved, it is necessary not only to specify the classes of data required by the templates in Clause 4 but also to achieve a common understanding of the semantics of this data. The FOM/SOM lexicon shall provide a means for federations to document the definitions of all terms utilized during construction of FOMs, and for individual federates to document the definitions of all terms provided in their SOMs.

Federations may want to develop additional views on FOM and/or SOM data besides simple term definitions and those explicitly defined by the OMT tables. The absence of additional data views in this document shall not constrain federation or simulation developers from defining whatever data views make sense for their applications. Rather, by providing federation/simulation developers maximum flexibility in this regard, libraries of reusable data views (and automated tools that support them) may be constructed and made available for general use in future applications.

5.2 Table formats

5.2.1 Object class definitions

This subclause shall describe the format for defining the object classes that are specified in a given FOM or SOM. A simple template that shall be used for describing this information is provided in Table 17. The first column of this table shall contain the names of all object classes described in the FOM or SOM, with the second column describing the semantics for that class. Object class names shall use dot notation as necessary to uniquely identify the object class being defined. Abstract, higher-level superclasses of instantiable subclasses shall be defined as such, along with their purpose in the object class hierarchy. In the case of object classes that can have direct instances (concrete classes) a description of the real-world entity the class is intended to represent shall be provided, along with any additional information required to clarify the semantics of the class (e.g., fidelity). Users may include the names of the attributes of the object class, and the interactions that the class can initiate or be affected by, in the textual description of that object class.

Table 17. Object class definitions

Object Class Definitions	
Term	**Definition**
<term name>	<term definition>
<term name>	<term definition>
. . .	. . .
<term name>	<term definition>

5.2.2 Interaction class definitions

This subclause shall describe the format for defining the interactions that can occur between object classes in the FOM, and interactions that can be published and/or reflected at the individual simulation level in the SOM. The structure that shall be used for describing this information is provided in Table 18. The first column of this table shall contain the name of each interaction class. Interaction class names shall use dot notation as necessary to uniquely identify the interaction class being defined. The second column shall provide sufficient descriptive information about the interaction class to ensure that the semantics are clearly understood. For abstract interaction classes, this information shall include the rationale for the use of the class in the interaction class hierarchy, and may include the list of lower-level subclasses that it supports. For publishable interaction classes, the definition shall include a description of the real-world event that the interaction class is intending to represent. The names of the initiating and receiving objects associated with the interaction, and the parametric information that shall be included with the interaction, may also be provided.

Table 18. Interaction class definitions

Interaction Class Definitions	
Term	**Definition**
<term name>	<term definition>
<term name>	<term definition>
. . .	. . .
<term name>	<term definition>

5.2.3 Attribute definitions

This subclause shall describe the format for defining the attributes that characterize HLA object classes. The structure that shall be used for describing this information is provided in Table 19. The first column of this table shall contain the name of the object class to which a given attribute belongs. This information is useful for associating attributes with object classes, but shall also be used to distinguish between attributes that share a common name but reside in different classes. Object class names shall use dot notation as necessary to uniquely identify the object class. The second column of this table shall contain the name of the attribute. The third column of this table shall describe the characteristic of the object class that this attribute is designed to measure. Characteristics of the attribute that are described in the OMT attribute table (units, resolution, update rate, etc.) may be repeated in the definition if it clarifies the meaning and purpose for the term.

Table 19. Attribute definitions

Attribute Definitions		
Class	**Term**	**Definition**
<term name>	<term name>	<term definition>
<term name>	<term name>	<term definition>
. . .	. . .	. . .
<term name>	<term name>	<term definition>

5.2.4 Parameter definitions

This subclause shall describe the format for defining the parameters that are associated with interaction classes. The structure that shall be used for describing this information is provided in Table 20. The first column of this table shall contain the name of the interaction class with which a given parameter is associated. Interaction class names shall use dot notation as necessary to uniquely identify the interaction class. The second column of this table shall contain the name of the parameter. The third column of this table shall describe the specific information that this parameter is designed to convey. Characteristics of the parameter that are described in the OMT parameter table (units, resolution, accuracy, etc.) may be repeated in the definition if it clarifies the meaning and purpose for the term.

Table 20. Parameter definitions

Parameter Definitions		
Class	**Term**	**Definition**
<term name>	<term name>	<term definition>
<term name>	<term name>	<term definition>
. . .	. . .	. . .
<term name>	<term name>	<term definition>

ANNEX A (informative)
Table entry notation

The OMT table specifications for the object class structure table, interaction class structure table, attribute table, and parameter table use a subset of Backus-Naur Form (BNF) [B2] to specify the types of entries that belong in particular table cells. In BNF, the types of terms to be substituted in the table are enclosed in angle brackets (e.g., *<class>*). Optional entries are enclosed in square brackets (e.g., *[(<ps>)]* for the optional *Publishable/Subscribable* capability entries of the object class structure table). Any parentheses are terminal characters that should appear as shown. Thus, the basic entry in a cell of the object class structure table, designated by *<class> (<ps>)*, indicates a class name followed by a *Publishable/Subscribable* code in parentheses. An asterisk (*) is used to indicate a repetition of zero or more instances, such as in the last column of the object class structure table where it indicates a variable number of entries for the most specific types of classes, as follows:

```
[<class> (<ps>)]  [,<class> (<ps>)]*  |  [<ref>]
```

A vertical bar (|) is used to indicate alternative possible entries. Thus, the specification for the last column of the object class structure table (above) indicates optional entries of either a variable-length list of classes with *Publishable/Subscribable* codes or a reference to another table.

ANNEX B (normative) Attribute/parameter basetypes

The following list shall define the complete set of basetypes that may be used to characterize object attributes or interaction parameters:

float - IEEE single-precision floating point number[B1]

double - IEEE double-precision floating point number[B1]

short - 16-bit, two's complement integer value in the range $-2^{15} \ldots 2^{15} - 1$

unsigned short - 16-bit, integer value in the range $0 \ldots 2^{16} - 1$

long - 32-bit, two's complement integer value in the range $-2^{31} \ldots 2^{31} - 1$

unsigned long - 32-bit, integer value in the range $0 \ldots 2^{32} - 1$

long long - 64-bit, two's complement integer value in the range $-2^{63} \ldots 2^{63} - 1$

unsigned long long - 64-bit, integer value in the range $0 \ldots 2^{64} - 1$

char - 8-bit quantity with a numerical value between 0 and 255 (decimal)

boolean - 1-bit quantity which can only take one of the values TRUE and FALSE

octet- 8-bit quantity guaranteed not to undergo any conversion

any - permits the specification of values which can express any basetype

string - one-dimensional array of "chars" which is terminated with a NULL (0 value) char

sequence - one-dimensional array of any basetype with two characteristics: a maximum size (which is fixed at specification time) and a length (which is determined at run time)

ANNEX C (informative)
Common normalization functions

The following normalization functions are commonly used and may be referred to directly in routing space tables. *RTI_min* and *RTI_max* are the RTI-specific minimum and maximum values for all routing spaces, and *RTI_range* = *RTI_max* - *RTI_min*. *Dimension_min* and *dimension_max* are the minimum and maximum possible values for the dimension as specified in the range element. For functions on enumerated values, *position_in_enumerated_set (domain)* is a function returning the domain value's position in the set element, starting at zero.

C.1 Linear

Form	linear(domain)
Parameters	domain: a non-enumerated value known to the federate using the routing space
Function	[(domain - dimension_min) / (dimension_max - dimension_min)] * (RTI_max - RTI_min) + RTI_min

C.2 Linear enumerated

Form	linear_enumerated(domain)
Parameters	domain: an enumerated value known to the federate using the routing space
Function	[position_in_enumerated_set(domain) /(enumerated_set_length - 1)] * (RTI_max - RTI_min)] / + RTI_min

C.3 Enumerated set

Form	enumerated_set(domain, mapped_set)
Parameters	domain: an enumerated value known to the federate using the routing space mapped_set: a list of values in the range [RTI_min, RTI_max] onto which there is a one-to-one mapping from the enumerated set
Function	mapped_set[position_in_enumerated_set(domain)]

C.4 Logarithmic

Form	logarithmic(domain)
Parameters	domain: a value known to the federate using the routing space
Function	[log(domain) * RTI_range]/log_range + RTI_min; where log_range = log(domain_max) - log(domain_min) = log(1E12) - log(1) = 12

ANNEX D (informative) Bibliography

[B1] IEEE Std 754-1985, Binary Floating-Point Arithmetic (ANSI).

[B2] Naur, P. et al., "Report on the Algorithmic Language ALGOL 60," *Communications of the ACM*, Vol. 6, No. 1, January 1963, pp. 1-17.

[B3] U.S. Department of Defense, Under Secretary of Defense (Acquisition and Technology) [USD (A&T)], *DoD Modeling and Simulation (M&S) Master Plan*, Washington, DC, October 1995.

[B4] U.S. Department of Defense, *High-Level Architecture, Data Distribution Management: Design Document Version 0.7*, November 1997.

[B5] U.S. Department of Defense, *High-Level Architecture, Federation Development and Execution Process Model Version 1.1*, December 1997.

[B6] U.S. Department of Defense, *High-Level Architecture, Glossary*, June 1997.

[B7] U.S. Department of Defense, *High-Level Architecture, OMT Test Procedures Version 1.1*, May 1997.

ANNEX E (normative)
OMT Data Interchange Format (DIF)

The High Level Architecture Object Model Template(HLA OMT) Data Interchange Format (DIF) is a standard file exchange format used to store and transfer HLA Federation Object Models (FOMs) and Simulation Object Models (SOMs) between FOM/SOM Builders. The DIF is built upon a common meta-model which represents the information needed to represent and manage object models created using the HLA OMT standard.

E.1 BNF Notation of the DIF

In order to ensure that there is no ambiguity in the definition of the DIF, the DIF has been formally defined in terms of Backus Naur Form (BNF). BNF is a formal notation used to describe inductive specifications. Attributed to John Backus and Peter Naur it was invented to describe the syntax of Algol 60 in an unambiguous manner. Since then it has become widely accepted and used by most authors of books on new programming languages to specify the syntax rules of the language.

Because no standard BNF notation exists, it is necessary to present the conventions for the notation used here. For the purposes of this document we will be using Extended BNF (EBNF) which includes some additional constructs to handle iteration and alternation as described in section E.1.2.

E.1.1 BNF Notation Conventions

BNF has three major parts:

- Terminals, which require no further definition,
- Non-terminals which are defined in terms of other non-terminals and terminals, and
- Productions which, for each non-terminal, precisely state how the non-terminal is constructed.

Certain symbols within the BNF have special meanings, these are called *meta-symbols* and they are used to structure the BNF. Double Quotes, Angle Brackets, Braces, etc. are meta-symbols within BNF, and their definition and use will be given below.

- Words inside double quotes ("word") represent literal words themselves (these are called *terminals*).
- Words contained within angle brackets '<>' represent semantic categories (i.e. *non-terminals*) which must be resolved by reading their definition elsewhere in the BNF. An example of a non-terminal is `<NameCharacter>`.
- A production (sometimes called a rule) is a statement of the definition of a non-terminal. It is designated by the production meta-symbol '::=' which assigns the definition to the right hand side (RHS) of the production to the non-terminal on the left hand side (LHS) of the production symbol. The LHS must always consist of a single non-terminal, while the RHS can consist of any combination of terminals and non-terminals. The symbol '`::=`' is read as "…is defined to be…" or "…is composed of…". An example of a production is: `<SpaceName> ::= <NameString>;`
- Selection of one item for a given instance is designated by use of the vertical bar symbol '|'. The symbol '|' is read as "…or…".
- Each BNF Statement is terminated by a semicolon (;).

E.1.2 EBNF Notation Conventions

- Terminals are represented using words inside double quotes. In addition, terminals will be further highlighted using **boldfaced** text. An example of a terminal is **`"HLA-FED"`**.
- The BNF used in this document adds a special case of non-terminal which is denoted by double brackets '<< >>' rather than single angle brackets. This special case non-terminal is a reference to the OMT DIF glossary found in section E.6.

- Optional Items are enclosed by square bracket meta-symbols '[' and ']'. Square brackets indicate the item exists either zero or one time, that is to say, it may or may not exist. An example of an optional item is `[<Sponsor>]` which indicates the Sponsor item may or may not be present in the DIF.
- Repetition (zero, one or many) is performed by the Curly Brace meta-symbols '{' and '}'.
  - Curly braces followed by a * character indicate that there are zero or more sequential instances of the item.
  - Curly braces followed by a + character indicate that there must be one or more sequential instances of the item.
- The double period .. used within a Literal is a shortcut notation for denoting the set of ASCII characters between the characters to either side of them. An example of this is `"a..z"` which denotes the set of lowercase letters between 'a' and 'z' inclusive.

E.1.3 Basic BNF Constructs

The following are a set of basic BNF constructs referenced in the main body of the DIF BNF, they are defined separately to make the main body more readable.

```
<Integer> ::= 0..65535;

<NameString> ::= <Letter> {<NameCharacter>}*;

<Letter> ::= "a..z" | "A..Z";
<Digit> ::= "0..9";
<NameCharacter> ::= <Letter> | <Digit> | "_" | "+" | "-" | "*" | "/" | "@" | "$" | "%" | "^" |
"&" | "=" | "<" | ">" | "~" | "!" | "#";
<TextString> ::= {<TextChar>}+;
<TextChar> ::= " "|"!"|"#..\'"|"*"|"+"|"-"|"."|"/"|"0..9"|":..@"|"A..Z"|"[..`"| "a..z"|"{..~";

<Date> ::= <Month> "/" <Day> "/" <Year>;
<Month> ::= ("01"|"02"|"03"|"04"|"05"|"06"|"07"|"08"|"09"|"10"|"11"|"12");
<Day> ::= (("0"|"1"|"2")<Digit>)|("30"|"31");
<Year> ::= <Digit><Digit><Digit><Digit>;

<NoteRef> ::= <OpenBracket> <RefList> <CloseBracket>;
<OpenBracket> ::= "[";
<RefList> ::= <RefNumber> {"," <RefNumber>}*;
<RefNumber> ::= "1..65535";
<CloseBracket> ::= "]";

<Identifier> ::= "0..4294967295"

<DataTypeName> ::= """ <BaseDataType> """ | <<DTP_Name>>;
<BaseDataType> ::= "unsigned short" | "short" | "unsigned long" | "long" | "unsigned long
                long" | "long long" | "double" | "float" | "boolean" | "any" | "string" | "char" |
                "octet";
```

`<NameString>` is a YACC compliant name which can consist of an initial letter followed by letters, digits, and any character in the set { "+", "-", "*", "/", "@", "$", "%", "^", "&", "=", "<", ">", "~", "!", "#"}. `<TextString>` is a string which does not contain double quotes, and can contain embedded "escaped" control codes using the C language conventions noted below. `<DataTypeName>` is either a reference to a base data type or the name of a defined `<DataType>`. Both base data types and defined data types will be enclosed with quotes. `<Identifier>` is a long Integer which is used to represent object and interaction class identity, and therefore must be unique. The `<Identifier>` has no inherent value other than to act as a representation of uniqueness.

Escape Character	Definition
'\n'	Newline
'\r'	Return
'\''	Single Quote
'\"'	Double Quote
'\\'	Backslash
'\t'	Tab
'\b'	Backspace

'\f'	FormFeed
'\xxx'	Any character where xxx is its Octal representation

E.2 HLA OMT DIF BNF Definition

```
 <HLA-OMT_DIF_v1.3> ::= <DIFHeader> {<ObjectModel>}+;

<DIFHeader> ::= "(DIF " <<DIFName>> <<DIFVerNum>> <DIFType>")";
<<DIFName>> ::= "HLA-OMT";
<<DIFVerNum>> ::= "v1.3";

<DIFType> ::=  "(TYPE " <<DIFTypeName>> ")";
<<DIFTypeName>> ::= "Single" | "Multiple" | "Directory";

<ObjectModel> ::= "(ObjectModel (Name " <<MOD_Name>> ")"
    "(VersionNumber " <<VER_Number>> ")"
        ["(Type " <<MOD_Type>> ")"]
        ["(Purpose " <<MOD_Purpose>> ")"]
        ["(ApplicationDomain " <<MOD_ApplicationDomain>> ")"]
        ["(SponsorOrgName " <<MOD_Sponsor_OrgName>> ")"]
        ["(POCHonorificName " <<MOD_POC_Honorific>> ")"]
        ["(POCFirstName " <<MOD_POC_FirstName>> ")"]
               ["(POCLastName " <<MOD_POC_LastName>> ")"]
               ["(POCOrgName " <<MOD_POC_OrgName>> ")"]
               ["(POCPhone " <<MOD_POC_Phone>> ")"]
               ["(POCEmail " <<MOD_POC_Email>> ")"]
               ["(ModificationDate " <<VER_ModificationDate>> ")"]
               ["(MOMVersion " <<MOD_MOMVersion>> ")"]
               {<OMTComponent>}* );

<<MOD_Name>> ::= """ <TextString> """;
<<VER_Number>> ::= """ <TextString> """;
<<MOD_Type>> ::= "FOM" | "SOM" | "OTHER";
<<MOD_Purpose>> ::= """ <TextString> """;
<<MOD_ApplicationDomain>> ::= """ <TextString> """;
<<MOD_Sponsor_OrgName>> ::= """ <TextString> """;
<<MOD_POC_Honorific>> ::= """ <TextString> """;
<<MOD_POC_FirstName>> ::= """ <TextString> """;
<<MOD_POC_LastName>> ::= """ <TextString> """;
<<MOD_POC_OrgName>> ::= """ <TextString> """;
<<MOD_POC_Phone>> ::= """ <TextString> """;
<<MOD_POC_Email>> ::= """ <TextString> """;
<<VER_ModificationDate>> ::= <Date>;
<<MOD_MOMVersion>> ::= """ <TextString> """;

<OMTComponent> ::= <Class>|<Interaction>|<DataType>|<Note>|<RoutingSpace>;
<Class> ::= "(Class (ID " <<CLS_ID>> ")"
    "(Name " <<CLS_Name>> [<NoteRef>] ")"
        ["(MOMClass " <<CLS_IsMOMClass>> ")"]
        "(PSCapbilities " <<CLS_PSCapbilities>> ")"
        ["(Description " <<CLS_Description>> ")"]
        {<ClassComponent>}* );

<<CLS_ID>> ::= <Identifier>;
<<CLS_PSCapbilities>> ::= "P" | "S" | "PS" | "N";
<<CLS_Name>> ::= """ <NameString> """;
<<CLS_IsMOMClass>> ::= "TRUE" | "FALSE";
<<CLS_Description>> ::= """ <TextString> """;

<ClassComponent> ::= <Attribute> | <SuperClass>;

<Attribute> ::= "(Attribute (Name " <<ATT_Name>> [<NoteRef>] ")"
    ["(DataType " <<ATT_DataType>> [<NoteRef>] ")"]
        ["(Cardinality " <<ATT_Cardinality>> [<NoteRef>] ")"]
        ["(Units " <<ATT_Units>> [<NoteRef>] ")"]
        ["(Resolution " <<ATT_Resolution>> [<NoteRef>] ")"]
        ["(Accuracy " <<ATT_Accuracy>> [<NoteRef>] ")"]
        ["(AccuracyCondition " <<ATT_AccuracyCondition>> [<NoteRef>] ")"]
        ["(UpdateType " <<ATT_UpdateType>> [<NoteRef>] ")"]
        ["(UpdateCondition " <<ATT_UpdateCondition>> [<NoteRef>] ")"]
        ["(TransferAccept " <<ATT_TransferAccept>> [<NoteRef>] ")"]
        ["(UpdateReflect " <<ATT_UpdateReflect>> [<NoteRef>] ")"]
        ["(Description " <<ATT_Description>> ")"] ;
        ["(RoutingSpace " <<ATT_RoutingSpace>> ")"] ")";

<<ATT_Name>> ::= """ <NameString> """;
<<ATT_DataType>> ::= <DataTypeName>;
<<ATT_Cardinality>> ::= """ <TextString> """;
<<ATT_Units>> ::= """ <TextString> """;
<<ATT_Resolution>> ::= """ <TextString> """;
<<ATT_Accuracy>> ::= """ <TextString> """;
<<ATT_AccuracyCondition>> ::= """ <TextString> """;
<<ATT_UpdateType>> ::= "Static" | "Periodic" | "Conditional";
<<ATT_UpdateCondition>> ::= """ <TextString> """;
<<ATT_TransferAccept>> ::= "T" | "A" | "TA" | "N";
<<ATT_UpdateReflect>> ::= "U" | "R" | "UR";
<<ATT_Description>> ::= """ <TextString> """;
<<ATT_RoutingSpace>> ::=  <<RSP_Name>> ;

<SuperClass> ::= "(SuperClass " <<CLS_ID>> ")";

<Interaction> ::= "(Interaction (ID " <<INT_ID>> ")"
    “(Name " <<INT_Name>> [<NoteRef>] ")"
        "(ISRType " <<INT_ISRType>> [<NoteRef>] ")"
        ["(MOMInteraction " <<INT_IsMOMInteraction>> ")"]
        ["(Description " <<INT_Description>> ")"]
        ["(RoutingSpace " <<INT_RoutingSpace>> ")"]
        {<InteractionComponent>}* );

<<INT_ID>> ::= <Identifier>;
<<INT_Name>> ::= """ <NameString> """;
<<INT_IsMOMInteraction>> ::= "TRUE" | "FALSE";
<<INT_ISRType>> ::= "I" | "S" | "R" | "IS" | "IR" | "N";
<<INT_Description>> ::= """ <TextString> """;
<<INT_RoutingSpace>> ::=  <<RSP_Name>> ;

<InteractionComponent> ::= <Parameter> | <SuperInteraction>;
<Parameter> ::= "(Parameter (Name " <<PRM_Name>> [<NoteRef>] ")"
    ["(DataType " <<PRM_DataType>> [<NoteRef>] ")"]
        ["(Cardinality " <<PRM_Cardinality>> [<NoteRef>] ")"]
        ["(Units " <<PRM_Units>> [<NoteRef>] ")"]
        ["(Resolution " <<PRM_Resolution>> [<NoteRef>] ")"]
        ["(Accuracy " <<PRM_Accuracy>> [<NoteRef>] ")"]
        ["(AccuracyCondition " <<PRM_AccuracyCondition>> [<NoteRef>] ")"]
        ["(Description " <<PRM_Description>> ")"] ")";

<<PRM_Name>> ::= """ <NameString> """;
<<PRM_DataType>> ::= <DataTypeName>;
<<PRM_Cardinality>> ::= """ <TextString> """;
<<PRM_Units>> ::= """ <TextString> """;
<<PRM_Resolution>> ::= """ <TextString> """;
<<PRM_Accuracy>> ::= """ <TextString> """;
<<PRM_AccuracyCondition>> ::= """ <TextString> """;
<<PRM_Description>> ::= """ <TextString> """;

<SuperInteraction> ::= "(SuperInteraction " <<INT_ID>> ")";

<DataType> ::= <EnumeratedDataType> | <ComplexDataType>;

<EnumeratedDataType> ::= "(EnumeratedDataType (Name " <<DTP_Name>> [<NoteRef>] ")"
     ["(MOMEnumeratedDataType " <<DTP_IsMOMDataType>> “)”]
     {<Enumeration>}+;

<<DTP_Name>> ::= """ <NameString> """;
<<DTP_IsMOMDataType>> ::= "TRUE" | "FALSE";

<Enumeration> ::= "(Enumeration (Enumerator " <<ENM_Enumerator>> [<NoteRef>] ")"
     "(Representation " <<ENM_Representation>> [<NoteRef>] ")" ")";
<<ENM_Enumerator>> ::= """ <NameString> """;
<<ENM_Representation>> ::= <Integer>;

<ComplexDataType> ::= "(ComplexDataType (Name " <<DTP_Name>> [<NoteRef>] ")"
     ["(MOMComplexDataType " <<DTP_IsMOMDataType>> ")"]
     {<ComplexComponent>}+;
<ComplexComponent ::= "(ComplexComponent (FieldName " <<CCP_FieldName>> [<NoteRef>] ")"
     ["(DataType " <<CCP_DataType>> [<NoteRef>] ")"]
     ["(Cardinality " <<CCP_Cardinality>> [<NoteRef>] ")"]
     ["(Units " <<CCP_Units>> [<NoteRef>] ")"]
     ["(Resolution " <<CCP_Resolution>> [<NoteRef>] ")"]
     ["(Accuracy " <<CCP_Accuracy>> [<NoteRef>] ")"]
     ["(AccuracyCondition " <<CCP_AccuracyCondition>> <NoteRef>] ")"] )";

<<CCP_FieldName>> ::= """ <NameString> """;
<<CCP_DataType>> ::= <DataTypeName>;
<<CCP_Cardinality>> ::= """ <TextString> """;
<<CCP_Units>> ::= """ <TextString> """;
<<CCP_Resolution>> ::= """ <TextString> """;
<<CCP_Accuracy>> ::= """ <TextString> """;
<<CCP_AccuracyCondition>> ::= """ <TextString> """;

<Note> ::= "(Note (NoteNumber " <<NC_NoteNumber>> ")"
     "(NoteText " <<NC_NoteText>>) ")";

<<NC_NoteNumber>> ::= <RefNumber>;
<<NC_NoteText>> ::= """ <TextString> """;

<RoutingSpace> ::= "(RoutingSpace (Name " <<RSP_Name>> [<NoteRef>] ")"
                      {<Dimension>}* ")";
<<RSP_Name>> ::= """ <NameString> """;

<Dimension> ::= "(Dimension (Name " <<DIM_Name>> [<NoteRef>] ")"
     "(DimensionType " <<DIM_DataType>> [<NoteRef>] ")"
     <DimensionDomain>
     ["(RangeSetUnits " <<DIM_Units>> [<NoteRef>] ")"]
     "(NormalizationFunction " <<DIM_NormalizationFunction>> [<NoteRef>]")" ")";

<<DIM_Name>> ::= """ <NameString> """;
<<DIM_DataType>> ::= <DataTypeName>;
<<DIM_Units>> ::= """ <TextString> """;
<<DIM_NormalizationFunction>> ::= """ <TextString> """;

<DimensionDomain> ::= <DimensionRange> | <DimensionSet>;

<DimensionRange> ::= "(DimensionMinimum " <<DIM_Minimum>> [<NoteRef>] ")"
     ["(DimensionMaximum " <<DIM_Maximum>>  ")"]
     "(IntervalType " <<DIM_IntervalType>> ")";

<<DIM_Minimum>> ::= """ <TextString> """;
<<DIM_Maximum>> ::= """ <TextString> """;
<<DIM_IntervalType>> ::= "Open" | "Closed";

<DimensionSet> ::= "(DimensionSet " {<DimensionSetMember>}+ [<NoteRef>]")";
<DimensionSetMember> ::= "(Member " <<DSM_Value>> ")";
<<DSM_Value>> ::= """ <TextString> """;
```

E.3 DIF Representation

This Data Interchange Format (DIF) has been structured as a stream of Object Model meta-data, and does not specify the specific physical representation or transport media used to exchange this meta-data. One possible representation and transport method is to use a simple ASCII File for the exchange of HLA OMT meta-data. The initial HLA object modeling tools will use this approach for the interchange of FOMs and SOMs, but this standard does not restrict the interchange to this single approach.

E.4 Types of DIF Instances

The BNF above defines a `<DIFType>` non-terminal whose purpose is to categorize an instance of the DIF into one of three types:

Single: A single model and version instance, containing the meta-data of a one version of a single model.

This is expected to be the most common use of the DIF, and is the standard unit of exchange between FOM Development Tools and Repositories.
Multiple: This keyword indicates a DIF instance where multiple versions and/or models can be included. This type of instance would be used to archive or backup a repository of OMT object models, or as a mechanism for exchanging sets of model/versions between project specific repositories and public repositories.
Directory: This keyword indicates a special type of DIF instance, one which is the response to the query of an object model repository for a "directory-listing" of the contents of the repository. This instance would contain only the Model and Version "clauses" which describe the Model and Version meta-data, but would not include the actual model content (i.e. the Classes, Interactions, data types, etc.). This provides the FOM Development Tool with the ability to present their user with a list of the contents of the repository, and the necessary meta-data with which to extract the selected model/version.

E.5 DIF Meta-Data Consistency

The use of BNF can not completely capture all of the rules that specify a complete and correct DIF file or object model. An OMT DIF file must comply with the following rules to be complete, consistent, and correct:

1. Each object model in a DIF file will contain at least one interaction class or one object class. An object model with only interaction classes or object classes is valid.
2. The identifier used to specify a parent object class (SuperClass) must be used in the same object model to identify a object class.
3. The identifier used to specify a parent interaction class (SuperInteraction) must be used in the same object model to identify a interaction class.
4. A data type specified for an attribute must be defined in the same object model if it is not one of the base data types.
5. A data type specified for an parameter must be defined in the same object model if it is not one of the base data types.
6. A data type specified for an complex data type field must be defined in the same object model if it is not one of the base data types.
7. A routing space specified for an attribute must be defined in the same object model.
8. A routing space specified for an interaction class must be defined in the same object model.
9. A data type specified for a routing space dimension must be defined in the same object model if it is not one of the base data types.
10. If an enumerated data type is specified as the data type for a routing space dimension, the dimension set must contain only enumerators from that enumerated data type.
11. Note numbers specified for an entry (e.g., object class name) in an object model must be defined within the same object model.
12. The same note number may be referenced by multiple entries in an object model.
13. All names (object class, attribute, etc.) are considered to be case insensitive.
14. Object class names must be unique where they share a common parent class. Object class names may be reused across multiple branches or tiers of the object class hierarchy, so long as no two sibling classes have the same name.
15. Interaction class names must be unique where they share a common parent class. Interaction class names may be reused across multiple branches or tiers of the interaction class hierarchy, so long as no two sibling classes have the same name.
16. Each object class in an object model must have a unique identifier in the DIF. Each interaction class identifier must be unique. It is valid to have an object class and interaction class identifiers with equivalent values.
17. Wherever a literal space appears in the DIF definition, multiple spaces are valid.
18. One or more literal spaces are allowed between any parentheses and the adjoining text.
19. Routing space names within an object model must be unique.
20. Dimension names within a single routing space must be unique
21. All terminals in the BNF description and DIF files produced in accordance with this BNF description are considered to be case insensitive. For example, the literal “ObjectModel” and “OBJECTMODEL” are

considered equivalent. Capitalization is used in the BNF strictly to enhance readability.

E.6 OMT DIF Glossary

This glossary defines the terms used in the HLA OMT DIF BNF definition to the corresponding concepts in the main body of the OMT document.

ATT_Accuracy	The optional specification, in TextString format, of the accuracy of an attribute.
ATT_AccuracyCondition	The optional specification, in TextString format, of the accuracy condition of an attribute.
ATT_Cardinality	The optional specification, in TextString format, of the cardinality of an attribute.
ATT_DataType	The optional user specification of a DataType.
ATT_Description	The optional specification, in TextString format, of the description of an attribute.
ATT_Name	The mandatory specification of an attribute name.
ATT_Resolution	The optional specification, in TextString format, of resolution of an attribute.
ATT_RoutingSpace	The optional specification, in NameString format, of a routing space name.
ATT_TransferAccept	The predetermined optional specification of the transfer acceptance of an attribute. The valid contents are “T”, “A”, “TA”, or “N”.
ATT_Units	The optional specification, in TextString format, of units of an attribute.
ATT_UpdateCondition	The optional specification, in TextString format, of the update condition of an attribute.
ATT_UpdateReflect	The predetermined optional specification of an update reflect. Valid contents are “U”,“R”, or “UR”.
ATT_UpdateType	The optional predetermined specification of the update type of an attribute. Valid contents are “Static”, “Periodic”, or “Conditional”.
CCP_Accuracy	The optional specification, in TextString format, of the accuracy of a complex component.
CCP_AccuracyCondition	The optional specification, in TextString format, of the accuracy condition of a complex component.
CCP_Cardinality	The optional specification, in TextString format, of the cardinality of a complex component.
CCP_DataType	The optional specification of a datatype name.
CCP_FieldName	The mandatory specification, in NameString format, of the name of a complex component.
CCP_Resolution	The optional specification, in TextString format, of the resolution of a complex component.
CCP_Units	The optional specification, in TextString format, of the unit of measure of a complex component.
CLS_Description	The optional description field of a class.
CLS_ID	The mandatory identifier that represents an instance of a class component.
CLS_IsMOMClass	The optional predetermined specification identifying whether a class is a MOM class. Valid contents are “True” or “False”.
CLS_Name	The mandatory name of a class component.
CLS_PSCapabilities	The mandatory specification of he publish/subscribe capabilities of the class. Valid contents are “P”, “S” or “PS”.
DIFName	The name of a DIF in TextString format.
DIFTypeName	An indicator of the contents of a DIF file, i.e, whether it contains a single object model, multiple object models, or a directory of object models. one of the following names to be assigned to a DIF Type,“Single" , "Multiple" , "Directory".
DIFVerNum	The identifier that represents a version number of a DIF in TextString format.
DIM_DataType	The mandatory specification of a datatype name.

DIM_IntervalType	The predetermined mandatory specification of the interval type of the dimension range. Valid values are “Open” or “Closed”.
DIM_Maximum	The optional specification, in TextString format, of the maximum value of a dimension range.
DIM_Minimum	The mandatory specification, in TextString format, of the minimum value in a dimension range.
DIM_Name	The mandatory specification, in NameString format, of the name of a Dimension.
DIM_NormalizationFunction	The specification, in TextString format, of the normalization functions of a dimension.
DIM_Units	The optional specification, in TextString format, of the units of measure of a dimension.
DSM_Value	The specification, in TextString format, of an iteration of a dimension set member.
DTP_IsMOMDataType	The optional predetermined specification of whether a parameter is a MOM interaction. Valid contents are “True” or “False”.
DTP_Name	The mandatory specification, in NameString format, of a datatype name
ENM_Enumerator	The mandatory specification, in NameString format, of the name of an enumerator.
ENM_Representation	The mandatory specification, in numerical format, of an enumeration representation.
INT_Description	The optional specification, in TextString format, of a description of an interaction.
INT_ID	The mandatory numerical specification of the instance of an interaction component.
INT_IsMOMInteraction	The optional predetermined specification of whether an interaction is a MOM interaction. Valid contents are “True” or “False”.
INT_ISRType	The mandatory predetermined specification of the type of ISR. Valid values are “I”, “S”, “R”, “IS”, “IR”, or “N”.
INT_Name	The mandatory specification, in NameString format, of the name of an interaction.
INT_RoutingSpace	The optional specification, in NameString format, of the name of a routing space.
MOD_ApplicationDomain	The text of the general type of application domain (e.g., Strike, AirDefense). Might be taken from categories of the Conceptual Models of the Mission Space (CMMS) when available. TextString format.
MOD_MOMVersion	The MOM version applicable to this object model version in TExtString format.
MOD_Name	A name for the federation or simulation
MOD_POC_Email	The email address, in TextString format, where the developing organization point of contact is located.
MOD_POC_FirstName	The first name, in TextString format, of the person designated as the point of contact by the developing organization who can handle inquires on operations and distribution.
MOD_POC_Honorific	The rank or title, in TextString format, of the person designated as the point of contact by the developing organization who can handle inquiries on operations and distribution.
MOD_POC_LastName	The last name, in TextString format, of the point of contact within the developing organization.
MOD_POC_OrgName	The organization name, in TextString format, of the developer organization point of contact.
MOD_POC_Phone	The phone number, in TextString format, of the point of contact within the developing organization.
MOD_Purpose	The text of the specific purpose for which the federation or simulation was created. Problem description should provide some notion of fidelity requirements for the given application. TextString format.
MOD_Sponsor_OrgName	The organization which sponsored the development of the simulation or federation. The name of the organization in TextString format, sponsoring development of the simulation or federation.
MOD_Type	The predefined specification of an object model. Valid contents are FOM, SOM, or OTHER.

NC_NoteNumber	The mandatory specification of a note number.
NC_NoteText	The mandatory specification, in TextString format, of the text of a note.
PRM_Accuracy	The optional specification, in TextString format, of the accuracy of a parameter.
PRM_AccuracyCondition	The optional specification, in TextString format, of the accuracy condition of a parameter.
PRM_Cardinality	The optional specification, in TextString format, of the cardinality of a parameter.
PRM_DataType	The optional specification of the datatype of a parameter.
PRM_Description	The optional specification, in TextString format, of the description of a parameter.
PRM_Name	The mandatory specification of a parameter name.
PRM_Resolution	The optional specification, in TextString format, of the resolution of a parameter.
PRM_Units	The optional specification, in TextString format, of the units of measure in a parameter.
RSP_Name	The mandatory specification, in NameString format, of a routing space name.
VER_ModificationDate	The date, in “Date” format, on which the accompanying FOM or SOM was last updated, to be the creation date if there have been no updates since then.
VER_Number	The version number assigned to an federation or simulation model.

E.7 Sample OMT DIF File

```
(DIF HLA-OMT v1.3 (TYPE Single))
(ObjectModel (Name "Restaurant SOM")
      (VersionNumber "1.0 Alpha")
      (Type SOM)
      (Purpose "To provide an example of an object model for a restaurant
federate")
      (ApplicationDomain "Restaurant operations")
      (SponsorOrgName "Federated Foods")
      (POCHonorificName "Mr.")
      (POCFirstName "Joseph")
      (POCLastName "Smith")
      (POCOrgName "Federated Foods")
      (POCPhone "(977) 555-1234")
      (POCEmail "smithj@fedfoods.com")
      (ModificationDate 01/01/1998)
      (MOMVersion "1.3")

(Class (ID 0)
      (Name "Customer")
      (PSCapabilities PS)
      (Description "Patron who wishes to eat a meal.")
)
(Class (ID 1)
      (Name "Bill")
      (PSCapabilities PS)
      (Description "Statement of money owed.")
)
(Class (ID 2)
      (Name "Order")
      (PSCapabilities PS)
      (Description "A specific meal a customer wants to purchase.")
)
(Class (ID 3)
      (Name "Employee")
      (PSCapabilities S)
      (Description "A person working for another person or business.")
      (Attribute (Name "Pay_Rate")
             (DataType "float")
             (Cardinality "1")
             (Units "Cents/Hour")
             (Resolution "1")
             (Accuracy "perfect")
             (AccuracyCondition "always")
             (UpdateType Conditional)
             (UpdateCondition "Merit Increase" [1])
             (TransferAccept TA)
             (UpdateReflect UR)
      )
       (Attribute (Name "Years_of_Service")
                (DataType "short")
                (Cardinality "1")
                (Units "Years")
                (Resolution "1")
                (Accuracy "Perfect")
                (AccuracyCondition "always")
                (UpdateType Periodic)
                (UpdateCondition "1/year, on Anniversary")
                (TransferAccept TA)
                (UpdateReflect UR)
       )
       (Attribute (Name "Home_Number")
                (DataType "string")
                (Cardinality "1")
                (Accuracy "perfect")
                (AccuracyCondition "always")
                (UpdateType Conditional)
                (UpdateCondition "Employee Request")
                (TransferAccept TA)
                (UpdateReflect UR)
       )
       (Attribute (Name "Home_Address")
                (DataType "Address_Type")
                (Cardinality "1")
                (UpdateType Conditional)
                (UpdateCondition "Employee Request")
                (TransferAccept TA)
                (UpdateReflect UR)
       )
    )
    (Class (ID 4)
        (Name "Dishwasher")
              (PSCapabilities PS)
              (Description "Person who cleans plates, pots, or utensils.")
             (SuperClass 3)
    )
    (Class (ID 5)
              (Name "Cook")
        (PSCapabilities PS)
             (Description "Person who prepares the meal.")
             (SuperClass 3)
    )
    (Class (ID 6)
        (Name "Food" )
        (PSCapabilities S)
        (Description "Nourishing substance that is eaten.")
    )
    (Class (ID 7)
        (Name "Main_Course")
        (PSCapabilities PS)
        (Description "The principal dish of a meal.")
          (SuperClass 6)
    )
    (Class (ID 8)
        (Name "Drink")
        (PSCapabilities S)
        (Description "Liquid that is taken into the mouth and swallowed.")
        (Attribute (Name "Number_Cups")
                (DataType "short")
                (Cardinality "1")
                (Units "Cups")
                (Resolution "1")
                (Accuracy "perfect")
                (AccuracyCondition "always")
                (UpdateType Conditional)
                (UpdateCondition "Customer Request")
                (TransferAccept TA)
                (UpdateReflect UR)
        )
        (SuperClass 6)
    )
    (Class (ID 9)
        (Name "Water")
        (PSCapabilities PS)
        (Description "A compound of hydrogen and oxygen.")
              (SuperClass 8)
     )
     (Class (ID 10)
        (Name "Coffee")
        (PSCapabilities PS)
        (Description "Beverage derived from coffee beans.")
              (SuperClass 8)
     )
     (Class (ID 11)
        (Name "Soda")
        (PSCapabilities PS)
        (Description "Carbonated beverage.")
        (Attribute (Name "Flavor")
                (DataType "Flavor_Type")
                (Cardinality "1")
                (UpdateType Conditional)
                (UpdateCondition "Customer Request")
                (TransferAccept TA)
                (UpdateReflect UR)
        )
                       (SuperClass 8)
     )
     (Class (ID 12)
        (Name "Appetizer")
        (PSCapabilities S)
        (Description "A portion of food or drink served before a meal.")
              (SuperClass 6)
     )
     (Class (ID 13)
        (Name "Soup")
        (PSCapabilities S)
        (Description "Liquid food made by boiling or simmering meat, fish, or
vegetables.")
```

```
              (SuperClass 12)
     )
     (Class (ID 14)
        (Name "Clam_Chowder")
        (PSCapabilities S)
        (Description "Thick soup made of clams or fish and vegetables.")
              (SuperClass 13)
     )
     (Class (ID 15)
        (Name "Manhattan")
        (PSCapabilities PS)
        (Description "Specific type of Clam Chowder made with salt pork and
tomatoes.")
              (SuperClass 14)
     )
     (Class (ID 16)
        (Name "New_England")
        (PSCapabilities PS)
        (Description "Specific type of Clam_Chowder made with onions and
potatoes.")
              (SuperClass 14)
     )
     (Class (ID 17)
        (Name "Beef_Barley")
        (PSCapabilities PS)
        (Description "A thin beef-flavored soup.")
              (SuperClass 13)
     )
     (Class (ID 18)
        (Name "Nachos")
        (PSCapabilities PS)
        (Description "A thin corn chip.")
              (SuperClass 12)
     )
     (Class (ID 19)
        (Name "Entree")
        (PSCapabilities S)
        (Description "The principal dish of a meal.")
              (SuperClass 6)
     )
     (Class (ID 20)
    (Name "Beef")
        (PSCapabilities PS)
        (Description "Flesh of a cow for use as meat.")
              (SuperClass 19)
     )
     (Class (ID 21)
        (Name "Chicken")
        (PSCapabilities PS)
        (Description "The flesh of domestic fowl.")
              (SuperClass 19)
     )
     (Class (ID 23)
        (Name "Seafood")
        (PSCapabilities S)
        (Description "Any salt-water fish or shellfish used for food.")
              (SuperClass 19)
     )
     (Class (ID 24)
        (Name "Fish")
        (PSCapabilities PS)
        (Description "Completely aquatic, cold-blooded vertebrates having
gills and fins.")
              (SuperClass 23)
     )
     (Class (ID 25)
        (Name "Shrimp")
        (PSCapabilities PS)
        (Description "Small, long-tailed crustacean.")
              (SuperClass 23)
     )
     (Class (ID 26)
        (Name "Lobster")
        (PSCapabilities PS)
        (Description "A marine, decapod crustacean.")
              (SuperClass 23)
     )
     (Class (ID 27)
              (Name "Pasta")
        (PSCapabilities PS)
        (Description "Various flour and egg food preparations.")
              (SuperClass 19)
     )
     (Class (ID 28)
        (Name "Side_Dish")
        (PSCapabilities S)
        (Description "An additional food item not considered part of the
Entree.")
              (SuperClass 6)
     )
     (Class (ID 29)
        (Name "Corn")
        (PSCapabilities PS)
        (Description "A vegetable consisting of the seeds of a cereal plant.")
              (SuperClass 28)
     )
     (Class (ID 30)
        (Name "Broccoli")
        (PSCapabilities PS)
        (Description "A vegetable consisting of branched greenish flower
heads.")
              (SuperClass 28)
     )
     (Class (ID 31)
        (Name "Baked_Potato")
        (PSCapabilities PS)
        (Description "A vegetable consisting of the tuber of the potato
plant.")
              (SuperClass 28)
     )
     (Class (ID 32)
        (Name "Dessert")
        (PSCapabilities S)
        (Description "Final course of a meal.")
              (SuperClass 6)
     )
     (Class (ID 33)
        (Name "Cake")
        (PSCapabilities PS)
        (Description "A sweet, baked, bread like food.")
```

```
            (SuperClass 32)
     )
     (Class (ID 34)
        (Name "Ice_Cream")
        (PSCapabilities S)
        (Description "A frozen food made of cream and variously flavored.")
              (SuperClass 32)
     )
     (Class (ID 35)
        (Name "Chocolate")
        (PSCapabilities PS)
        (Description "A preparation of the seeds of cacao.")
              (SuperClass 34)
     )
     (Class (ID 36)
        (Name "Vanilla")
        (PSCapabilities PS)
        (Description "Flavoring extract from the Vanilla plant.")
              (SuperClass 34)
     )
     (Class (ID 37)
        (Name "Waiter")
        (PSCapabilities PS)
        (Description "The person who waits on a table.")
        (Attribute (Name "Efficiency")
              (DataType "short")
                (Cardinality "1")
                (Resolution "1")
                (Accuracy "perfect")
                (AccuracyCondition "always")
                (UpdateType Periodic)
                (UpdateCondition "Performance Review")
                (TransferAccept TA)
                (UpdateReflect UR)
        )
        (Attribute (Name "Cheerfulness")
              (DataType "short")
                (Cardinality "1")
                (Resolution "1")
                (Accuracy "perfect")
                (AccuracyCondition "always")
                (UpdateType Periodic)
                (UpdateCondition "Performance Review")
                (TransferAccept TA)
                (UpdateReflect UR)
        )
        (Attribute (Name "State")
              (DataType "Waiter_Tasks")
                (Cardinality "1")
                (UpdateType Conditional)
                (UpdateCondition "Work Flow")
                (TransferAccept TA)
                (UpdateReflect UR)
        )
                    (SuperClass 3)
     )
     (Class (ID 38)
        (Name "Cashier")
        (PSCapabilities PS)
        (Description "The person who collects payments for purchases.")
```

```
              (SuperClass 3)
     )
     (Class (ID 39)
        (Name "Greeter")
        (PSCapabilities PS)
        (Description "A person who welcomes customers to a business
establishment.")
              (SuperClass 3)
     )
     (Interaction (ID 100)
              (Name "Customer_Employee_Transactions")
        (ISRType I)
        (Description "The base class of interactions between customers and
employee.")
     )
     (Interaction (ID 101)
        (Name "Customer_Seated")
        (ISRType IS)
               (SuperInteraction  100)
     )
     (Interaction (ID 102)
        (Name "Order_Taken")
        (ISRType I)
        (Description "The placement of a food order between waiter and
customer.")
              (SuperInteraction 100)
     )
     (Interaction (ID 103)
        (Name "Food_Served")
        (ISRType I)
        (Description "The serving of food between waiter and customer.")
              (SuperInteraction 100)
     )
     (Interaction (ID 104)
        (Name "Customer_Pays")
        (ISRType I)
        (Description "The action of paying for food and service.")
              (SuperInteraction 100)
     )
     (Interaction (ID 105)
             (Name "Customer_Leaves")
        (ISRType IS)
        (Description "The action of customer leaving the dining
establishment.")
              (SuperInteraction 100)
     )
     (Interaction (ID 106)
        (Name "Main_Course_Served")
        (ISRType I)
        (Description "Indicates food has passed from waiter to customer.")
        (Parameter (Name "Temperature_OK")
              (DataType "Temp_Type")
              (Cardinality "1")
        )
        (Parameter (Name "Accuracy_OK")
           (DataType "Accur_Type")
           (Cardinality "1")
        )
        (Parameter (Name "Timeliness_OK")
           (DataType "boolean")
           (Cardinality "1")
           (Resolution "1")
           (Accuracy "perfect")
           (AccuracyCondition "always")
        )
                        (SuperInteraction 103  )
     )
     (Interaction (ID 107)
        (Name "Order_Taken_From_Kids_Menu")
        (ISRType I)
        (Description "Indicates food order selection is from the children\'s
menu.")
            (SuperInteraction 102 )
     )
     (Interaction (ID 108)
        (Name "Order_Taken_From_Adult_Menu")
        (ISRType I)
        (Description "Indicates food order selection is from adult\'s menu.")
              (SuperInteraction 102)
     )
     (Interaction (ID 109)
        (Name "Drink_Served")
        (ISRType I)
        (Description "Indicates drink has passed from waiter to customer.")
              (SuperInteraction 103)
     )
     (Interaction (ID 110)
        (Name "Appetizer_Served")
        (ISRType I)
        (Description "Indicates the appetizer has passed from waiter to
customer.")
           (SuperInteraction 103)
     )
     (Interaction (ID 111)
        (Name "Dessert_Served")
        (ISRType I)
        (Description "Indicates dessert has passed from waiter to customer.")
           (SuperInteraction 103)
     )
     (Interaction (ID 112)
        (Name "Pay_Bill_by_Credit_Card")
        (ISRType I)
        (Description "Indicates payment for food and service is being made by
credit card.")
            (SuperInteraction 104)
     )
     (Interaction (ID 113)
        (Name "Pay_Bill_by_Cash")
        (ISRType I)
        (Description "Indicates payment for food and service is being made by
cash.")
            (SuperInteraction 104)
     )

     (EnumeratedDataType (Name "Waiter_Tasks")
           (Enumeration (Enumerator "Taking_Order")
           (Representation 1))

           (Enumeration (Enumerator "Serving")
           (Representation 2))
```

```
            (Enumeration (Enumerator "Cleaning")
            (Representation 3))

            (Enumeration (Enumerator "Calculating_Bill")
            (Representation 4))

            (Enumeration (Enumerator "Other")
            (Representation 5))
      )

      (EnumeratedDataType (Name "Flavor_Type")
         (Enumeration (Enumerator "Cola")
         (Representation 101))

         (Enumeration (Enumerator "Orange")
         (Representation 102))

         (Enumeration (Enumerator "Root_Beer")
         (Representation 103))

         (Enumeration (Enumerator "Cream")
         (Representation 104))
      )

      (ComplexDataType (Name "Address_Type")
         (ComplexComponent (FieldName "Street")
            (DataType "string")
            (Cardinality "1")
            (Units "N/A")
            (Accuracy "perfect")
            (AccuracyCondition "always")
      )
      (ComplexComponent (FieldName "City")
            (DataType "string")
            (Cardinality "1")
            (Units "N/A")
            (Accuracy "perfect")
            (AccuracyCondition "always")
      )
      (ComplexComponent (FieldName "State")
            (DataType "string")
            (Cardinality "1")
            (Units "N/A")
            (Accuracy "perfect")
            (AccuracyCondition "always")
      )
      (ComplexComponent (FieldName "Zip")
            (DataType "string")
            (Cardinality "1")
            (Units "N/A")
            (Accuracy "perfect")
            (AccuracyCondition "always")
      )
      )
      (ComplexDataType (Name "Temp_Type")
         (ComplexComponent (FieldName "Appetizer")
            (DataType "boolean")
            (Cardinality "1")
            (Units "N/A")
```

```
           (Resolution "1")
           (Accuracy "perfect")
           (AccuracyCondition "always")
        )
        (ComplexComponent (FieldName "Entree")
           (DataType "boolean")
           (Cardinality "1")
           (Units "N/A")
           (Resolution "1")
           (Accuracy "perfect")
           (AccuracyCondition "always")
        )
        (ComplexComponent (FieldName "Side_Dish")
           (DataType "boolean")
           (Cardinality "1")
           (Units "N/A")
           (Resolution "1")
           (Accuracy "perfect")
           (AccuracyCondition "always")
        )
     )
     (ComplexDataType (Name "Accur_Type")
        (ComplexComponent (FieldName "Drink")
           (DataType "boolean")
           (Cardinality "1")
           (Units "N/A")
           (Resolution "1")
           (Accuracy "perfect")
           (AccuracyCondition "always")
        )
        (ComplexComponent (FieldName "Appetizer")
           (DataType "boolean")
           (Cardinality "1")
           (Units "N/A")
           (Resolution "1")
           (Accuracy "perfect")
           (AccuracyCondition "always")
        )
        (ComplexComponent (FieldName "Entree")
           (DataType "boolean")
           (Cardinality "1")
           (Units "N/A")
           (Resolution "1")
           (Accuracy "perfect")
           (AccuracyCondition "always")
        )
        (ComplexComponent (FieldName "Side_Dish")
           (DataType "boolean")
           (Cardinality "1")
           (Units "N/A")
           (Resolution "1")
           (Accuracy "perfect")
           (AccuracyCondition "always")
        )
     )
(RoutingSpace (Name "Bar_Order")
      (Dimension (Name "Soda_Flavor")
            (DimensionType "Flavor_Type")
              (DimensionSet
                   (Member "Cola")
                   (Member "Orange")
                   (Member "Root_Beer") )
              (NormalizationFunction "Linear_enumerated (soda_flavor)")
      )
      (Dimension (Name "Bar_Quantity")
            (DimensionType "Short")
              (DimensionMinimum "1")
              (DimensionMaximum "25")
                (IntervalType Closed)
              (NormalizationFunction "Linear (number_sodas)")
           )
)
(RoutingSpace (Name "Server_Order")
      (Dimension (Name "Waiter_ID")
      (DimensionType "Short")
      (DimensionMinimum "1")
      (DimensionMaximum "20")
      (IntervalType Closed)
      (NormalizationFunction "Linear (waiter_ID)")
           )
)

(Note (NoteNumber 1)
        (NoteText "Merit raises are not provided according to any regular time
interval, but are provided on a supervisor\'s recommendation based on evidence
of exceptional performance.")))
```